Un langage de commande pour un logiciel extensible interactif de modélisation économétrique

Pascale Lespinasse-Hamoniaux

CONSERVATOIRE NATIONAL DES ARTS ET METIERS-

CENTRE AGREE
DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

LE DIPLÔME D'INGÉNIEUR C.N.A.M.

en

Informatique

par

Pascale LESPINASSE-HAMONIAUX

UN LANGAGE DE COMMANDE POUR UN LOGICIEL EXTENSIBLE INTERACTIF DE MODÉLISATION ÉCONOMÉTRIQUE

SOUTENU LE : 28 mai 1985.

JURY

Président : J.Y. RANCHIN

Membres : L. BOLLIET
R. RECHENMANN
M. MORINIERE

Je tiens à remercier,

Monsieur le Professeur J.Y. RANCHIN,
du Conservatoire National des Arts et Métiers qui a bien voulu présider le Jury de ce mémoire,

Monsieur le Professeur L. BOLLIET,
Directeur du GIS de Mini et Micro Informatique de Grenoble et Professeur à l'Université de Grenoble 2, pour l'honneur qu'il me fait de participer à ce Jury,

Monsieur F. RECHENMANN,
Ingénieur de Recherche à l'INRIA qui ne m'a pas ménagé ses encouragements et qui m'a conseillé tout au long de mon travail,

Monsieur M. MORINIERE,
de la Direction de la Prévision du Ministère des Finances qui a accepté d'être membre de mon Jury.

SOMMAIRE

V. Réalisations techniques et choix d'implémentation

V.1 Représentation interne d'une commande
V.2 Mémorisation d'une commande
V.3 La table des commandes
V.4 L'appel d'une commande
V.5 L'interpréteur du langage de commande

Conclusion

INTRODUCTION

En économétrie et notamment en modélisation, le recours a l'outil informatique est devenu indispensable. En effet les modèles atteignent des tailles de plus en plus grandes. De plus leur construction et leur utilisation nécessitent des calculs et des manipulations de données que seul l'ordinateur peut effectuer.

De nombreux logiciels ont été développés pour traiter tel ou tel aspect de la modélisation. Mais la définition d'un logiciel présentant à la fois les propriétés de complétude, d'extensibilité et de facilités d'utilisation n'a pas reçu jusqu'à présent toute l'attention qu'elle mérite.

Dans le cadre du développement d'un logiciel de modélisation macroéconomique, MODULECO, nous nous intéressons plus spécialement à l'interaction utilisateur-système par le biais d'un langage de commandes. Le travail décrit dans ce mémoire a concerné essentiellement la spécification et la réalisation de l'environnement du langage de commande de façon à assurer au logiciel des qualités d'extensibilité et de facilité de mise à jour. Ce logiciel étant destiné à être utilisé par des non informaticiens nous avons insisté sur l'aide fournie aux utilisateurs en cours de session sous la forme de documentation interactive. De plus le logiciel a été conçu pour prendre le maximum d'initiatives et pour éviter à l'utilisateur tous les travaux non économiques ou qui ne se rapportent pas directement à son étude.

Aprés avoir défini dans le premier chapitre le domaine d'application, on présente dans le deuxième chapitre les différents logiciels existants et les caractéristiques de MODULECO par rapport à ces derniers. Le troisième chapitre explique ce qu'est une commande et met en évidence les différents rôles du langage de commande de MODULECO. On étudie plus précisément dans le quatrième chapitre les différents mécanismes d'extension et de modification. Enfin le cinquième chapitre présente les solutions techniques retenues et leurs avantages.

MODULECO est un logiciel de construction et d'utilisation de modèles macroéconomiques. Il a été développé dans le cadre d'une association à but non lucratif, le Club MODULECO, créé en Février 1979.

La réalisation de ce logiciel a bénéficié du support de l'Agence de l'Informatique, de l'Assemblée Nationale, du Centre Interuniversitaire de Calcul de Grenoble, de la Direction de la Prévision du Ministère de l'Economie et des Finances, de l'I.N.R.I.A, de l'I.N.S.E.E, du Laboratoire I.M.A.G et du Sénat. L'O.F.C.E a rejoint plus récemment le Club MODULECO.

CHAPITRE I

LA MODELISATION MACROECONOMIQUE

1. Aperçu historique

2. Notion de modèle économétrique

3. Mode de travail de l'économètre

4. Les problèmes mathématiques et informatiques soulevés

1. Aperçu historique

La modélisation macroéconomique s'est imposée trés largement dans la pratique comme un instrument indispensable dans l'analyse économique empirique, la prévision économique et l'aide à l'élaboration des décisions de politique économique.

Dans un premier temps, la modélisation est considérée comme technique d'illustration d'une approche théorique. Dans cette optique, les modèles sont utilisés en simulation sur le passé pour confirmer la validité de la théorie économique qu'ils illustrent. Ensuite le modèle devient avant tout un outil d'aide à la prise de décisions en matière de politique économique. Le domaine d'application essentiel est la prévision à court-terme. Ce mouvement est renforcé par l'apparition des modèles trimestriels et par la construction des premiers modèles de grande taille. Pour rechercher des politiques économiques optimales ou simplement pour étudier les propriétés de stabilité des modèles et de la réalité économique qu'ils décrivent, on utilise les méthodes formelles d'optimisation.

La modélisation s'est diversifiée également dans les fins poursuivies. On a vu apparaître des modèles plus spécifiquement orientés vers l'étude de l'évolution à moyen et long terme, voire à trés long terme. En ce qui concerne les domaines économiques couverts, à côté des modèles représentant l'ensemble d'une économie, se sont développés des modèles partiels. Ils portent sur un secteur productif, un agent économique ou une région par exemple. A l'inverse des modèles multirégionaux, multinationaux, voire mondiaux ont été développés.

2. Notion de modèle économétrique

Quelques définitions

Les *agents économiques* sont les individus, les groupes d'individus ou les organismes qui constituent des centres de décisions et d'actions élémentaires. La *micro-économonomie* s'intéresse aux arbitrages effectués par les agents économiques entre les différents biens et services sur les différents marchés, dans un cadre national ou international, en ne considérant que le présent ou en prenant en compte le présent ou le futur. Le terme "micro" souligne que l'on considère les agents un par un.

Dans une approche *macroéconomique*, au contraire, on part de l'ensemble de l'économie, que l'on examine globalement, les agents étant considérés en groupe. Ces partenaires de l'activité économique ayant une même activité principale sont regroupés en sept *secteurs institutionnels* dans le cadre de la *comptabilité nationale*.

"La comptabilité nationale est la présentation suivant un cadre comptable rigoureux, de l'ensemble des informations chiffrées relatives à l'activité économique de la nation. Elle enregistre les transactions entre les grands ensembles qui constituent la nation." (E.Mallinvaud)

Cette activité peut être représentée à l'aide de *comptes*. Pour un secteur donné on distingue six sous-comptes principaux. Leurs soldes constituent les *agrégats* fondamentaux de l'économie (Produit Intérieur Brut, Produit National Brut, Revenu Intérieur Brut ...). La somme d'un agrégat donné pour les différents secteurs institutionnels donne la valeur de cette agrégat au niveau national.

Un modèle cherche à schématiser le fonctionnement de l'économie, à l'intérieur des limitations de champ que l'on s'est fixé, par un ensemble d'équations dont la solution est supposée décrire le phénomène étudié. Dans MODULECO, les modèles sont mis en oeuvre en temps discret sous la forme d'équations récurrentes.

Un modèle est donc la formalisation sous la forme d'un système d'équation des liens qui peuvent exister entre plusieurs agents économiques.

Les variables intervenant dans ces équations représentent des agrégats de l'économie. Elles sont considérées à l'instant t,période courante ou à l'instant t-i, périodes précédentes. Une variable endogéne est calculée par le modèle. Une variale exogène est une variable dont les valeurs historiques alimentent le modèle.

Une équation définit une variable expliquée à l'aide d'un certain nombre de variables explicatives, qui peuvent être d'autres variables endogènes du modèle ou des variables exogènes, de coefficients numériques inconnus et d'un terme résiduel aléatoire. Les différentes équations du modèle font intervenir des variables portant sur la période courante et des variables (endogènes et exogènes) retardées, c'est à dire portant sur la ou les périodes précédentes.

Formellement, un modèle économétrique s'écrit:

$$\Phi\ (Y_t\ ,\ Y_{t-1},\ \ldots,\ Y_{t-p},\ X_t,\ X_{t-1},\ \ldots,\ X_{t-q},\ \alpha,\ e) = 0$$

où Y est le vecteur des variables endogènes.

Y_t désigne le vecteur constitué d'observations, une pour chaque variable endogène à l'instant t. Y_{t-1} est le vecteur des variables endogènes retardées d'une période.

X est le vecteur des variables exogènes.

α est le vecteurs des coefficients.

e est le vecteur des termes résiduels ou variables d'écart. Elles permettent de caler le modèle.

Pour chaque variable du modèle, l'ensemble des observations passées constitue une série chronologique, c'est à dire une suite de valeurs réelles dans le temps. Outre le temps, une série chronologique peut avoir plusieurs dimensions. A chaque série est associé un calendrier qui définit le domaine temporel de la série et donc sa périodicité: annuel, trimestriel, mensuel, hebdomadaire, quotidien ou même quelconque (calendrier budgétaire de 15 mois par exemple).

Les modèles macroéconomiques ne s'intéressent pas à des comportements individuels (de ménages ou d'entreprises). Ils représentent le fonctionnement du système économique saisi au niveau agrégé.

3. Mode de travail de l'économètre

La première phase est la construction d'un ensemble d'équations. Une première forme d'équation est obtenue à partir d'analyses de séries et de recherches de variables explicatives.

Pour chaque équation, les coefficients sont estimés avec une méthode appropriée en utilisant un ensemble d'observations pour les variables expliquées et explicatives. Des statistiques sont interprétées pour déceler des problèmes (multicolinéarité, hétéroscédasticité, autocorrélation des termes d'erreurs) et pour juger de la qualité d'une estimation ("F" de Fisher, "t" de Student, R2). L'analyse des résidus, différence entre la valeur estimée d'une variable et sa valeur historique permet d'éliminer les représentations indésirables. Les résultats de ces estimations des coefficients peuvent amener à reformuler certaines équations.

Pour chaque équation, les principales étapes de cette phase sont itérées jusqu'à l'obtention d'un modèle satisfaisant pour la période d'estimation.

La deuxième phase consiste en la validation du modèle dans son ensemble

par une simulation sur le passé qui peut être étendue sur le futur. Les équations qui ne satisfont pas la validation du modèle sont reformulées. Le processus décrit jusqu'ici est itéré jusqu'à l'obtention d'un système d'équations satisfaisant.

4.Les problèmes mathématiques et informatiques rencontrés

Ampleur des calculs, volume des données

La construction, l'étude et l'utilisation de modèles de taille moyenne ou grande obligent à manipuler un grand nombre de données et à effectuer des calculs de grande ampleur. Ces tâches nécessitent donc un recours intensif à l'outil informatique.

Nombre de méthodes proposées

Ces tâches utilisent également un grand nombre de méthodes mathématiques adaptées aux divers problèmes soulevés: estimation, simulation et optimisation. La mise en oeuvre de ces méthodes doit être assurée par des algorithmes informatiques performants.

Rigidité du modèle et du mode de travail

Les premières générations de modèles ont souvent été écrits sous la forme d'un programme en langage évolué, comprenant simultanément les instructions relatives :

-à la formulation des équations,

-à l'algorithme de résolution ou d'optimisation,

-au test de convergence du modèle,

-à l'édition des résultats,

-au mode d'accés aux données.

En pratique ceci entraine une grande rigidité dans la représentation du modèle et des méthodes de résolution ou d'optimisation utilisées. Ceci nécessite aussi la réécriture, pour chaque application spécifique, des algorithmes nécessaires. Cette écriture est peu lisible et les mises à jour sont impossibles.

Conclusion

Un logiciel permettant de mettre en oeuvre les différents algorithmes de calcul et méthodes mathématiques, ainsi que certains utilitaires nécessaires à l'élaboration et au maniement des modèles est un outil extrémement précieux. Les fonctions qu'il doit remplir correspondent aux différentes étapes d'une modélisation économétrique.

De tels logiciels existent et quelques uns d'entre eux sont présentés dans le chapitre suivant.

CHAPITRE II

LES LOGICIELS DE TRAITEMENT DES MODELES.

1. Définition du logiciel idéal

2. Etude comparative de quelques logiciels existants

3. Caractéristiques de MODULECO

4. Présentation d'une étude économétrique à l'aide de MODULECO

1. Définition du logiciel idéal

Les besoins de l'économètre que nous avons mis en évidence définissent les caractéristiques que devrait posséder le logiciel idéal de modélisation économétrique.

Commodité et souplesse d'emploi

Il présente la double caractéristique d'une structure modulaire doublée d'une homogénéité de mise en oeuvre par l'utilisateur à travers un langage d'interaction dans lequel chaque instruction permet de déclencher toute une séquence de fonctions élémentaires. Au cours d'une utilisation interactive, une assistance est fournie à l'utilisateur. Celui ci peut à tout moment obtenir les renseignements dont il a besoin sur le mode d'emploi du logiciel.

Simplicité d'utilisation

Le langage d'interaction est suffisament simple pour pouvoir être utilisé par l'économiste n'ayant aucune connaissance informatique. Il est concis; le logiciel est conçu pour prendre le maximum d'initiatives et demander le minimum d'information à l'utilisateur.

Possibilité d'extension

Le langage d'interaction offre à l'utilisateur un moyen aisé pour intervenir sur l'écriture de son modèle et son estimation ou sur les séries. Pour répondre à des besoins spécifiques, l'utilisateur a la possibilité de créer de nouvelles commandes pouvant représenter une composition de commandes existantes. La structure modulaire permet un aménagement aisé du logiciel par adjonction ou remplacement de fonctions élémentaires, en fonction de l'évolution des besoins et des connaissances.

Mode interactif ou traitement par lots

L'utilisateur a l'initiative de faire appel aux différentes fonctionnalités du logiciel dans l'ordre qu'il juge raisonnable à la vue des résultats obtenus auparavant. Inversement, pour certaines tâches telle que la construction de la base de données, il est souhaitable de pouvoir utiliser le logiciel en traitement par lots.

Intégration du système

Il forme un système unique regroupant l'ensemble des sous-systèmes spécifiques à chacune des tâches économiques. Nous avons vu qu'une étude économétrique implique des itérations; la progression du spécialiste est incertaine et comporte des retours en arrière dans l'analyse. L'économètre doit donc disposer d'un logiciel intégré, lui permettant de passer facilement et rapidement d'une étape à une autre. Les résultats d'une tâche peuvent être utilisées par une autre tâche sans difficultés. De plus cette transmission étant sous le contrôle du système, elle présente toutes les garanties de sécurité.

Complétude

Sa complétude assure au spécialiste qu'il y trouvera tous les outils dont il a besoin. En particulier le sous-système d'estimation sera riche en méthodes.

2. Etude comparative de quelques logiciels existants

Parmi tous les logiciels de modélisation économétrique existants, nous n'en avons retenu que quelques uns de ceux qui sont munis d'un langage de commande.

Le système XING

Les logiciels XING et APACHE ont été développés par la Direction de la Prévision du Ministère des Finances. Ils fonctionnent sur trois matériels différents: PHILIPS P880, IBM série 370 et HONEYWELL-BULL, série 66. APACHE assure la gestion, la transformation des données et l'économétrie. Il fonctionne indifféremment en traitement par lots et en interactif. XING assure la résolution des modèles macroéconomiques et fonctionne en traitement par lots uniquement.

Le langage de commande de XING a une syntaxe simple dont l'apprentissage est facile. Cependant l'utilisateur n'a pas la possibilité d'étendre le langage. Il existe deux sortes de commandes: les commandes principales et les sous-commandes. Les commandes principales débutent une tâche. Les sous-commandes précisent des paramètres ou un traitement particulier dans la tâche.

En utilisation interactive, aucune aide n'est fournie à l'utilisateur.

XING et APACHE étant deux logiciels indépendants, l'utilisateur qui a effectué une simulation dans XING, doit en sauvegarder les résultats, sortir de XING et utiliser APACHE pour les analyser.

Le logiciel ISISMA

Il a été développé par le DDA (Danish Data Archives) et par l'Institut Economique (Université de Aarhus, Danemark).

La syntaxe du langage de commandes est simple, il est en format libre et il offre la possibilité d'employer des abréviations. ISISMA peut être utilisé en interactif comme en traitement par lots. L'utilisateur peut ajouter facilement ses propres algorithmes au logiciel. Celui-ci propose une macro "for loop" qui permet d'itérer un ensemble de commandes. Un éditeur permet de corriger un appel de commande erroné sans avoir à le retaper entièrement.

ISISMA est un ensemble de tâches réparties en six groupes. Chaque tâche est constituée d'un ensemble de commandes. Par exemple le groupe 2 concerne les fichiers. Il existe une tâche pour chaque type de fichiers; elle regroupe les

commandes de création, mise à jour et destruction des éléments d'un fichier. Il existe trois types de fichiers pour l'utilisateur: séries chronologiques, matrices et modèles. On peut également imprimer le contenu d'un fichier, lire ou copier des éléments d'un autre fichier.

Le logiciel T.S.P

T.S.P (Time Series Processor) est maintenu par la Computer Sciences Corporation en Californie. La version distribuée a été écrite pour un ordinateur IBM 360/370. Programmé dans un FORTRAN relativement standard, il est portable. Mais la conversion nécessaire à l'emploi sur une autre machine est à la charge de l'utilisateur.

A la différence des logiciels passés en revue jusqu'à maintenant, TSP est un "package", c'est à dire un ensemble de routines que l'utilisateur doit composer lui même en construisant des programmes qui les appellent. De plus il n'est pas interactif.

Il est muni d'un langage de haut niveau qui s'écrit en format libre. L'exécution d'un programme TSP comprend grossiérement trois phases:

1. interprétation et traduction de l'ensemble des instructions qui ne concernent pas le chargement des données. Le code objet résultant de la traduction est stocké en mémoire.

2. exécution séquentielle des instructions traduites avec branchement à la phase 3 quand l'instruction provoque le chargement des données.

3. Tant qu'il y a des instructions de chargement de données, interprétation, traduction et exécution de ces instructions.

TSP est facile à utiliser et d'un apprentissage aisé et rapide.

Il ne parait pas possible a priori d'étendre ce logiciel par ajout de nouveaux algorithmes.

Si nous citons ce logiciel, un peu loin du logiciel idéal décrit précédemment c'est qu'il est et qu'il reste encore le ou l'un des logiciels les plus répandus.

Le système E.P.S

E.P.S (Econometric Progamming System) a été développé par Data Resources Inc. C'est un système intégré d'aide à l'économiste dans toutes les phases de son analyse.

Un ensemble d'options dites globales pouvant être positionnées en dehors de toutes commandes est mis à la disposition de l'utilisateur. Ceci facilite beaucoup l'utilisation du logiciel.

Pour les travaux de grande taille ou itératifs, on peut utiliser des fichiers d'entrée contenant une suite de commandes. On peut passer d'un mode de travail à l'autre sans problèmes au cours d'une même session. Dans les fichiers d'entrée, on peut utiliser des instructions conditionelles. L'utilisateur a la possibilité de définir des routines en assemblant des appels à des commandes EPS à l'aide d'un macrolangage.

Avec TSP, c'est l'un des logiciels de modélisation macroéconomique le plus connu et le plus répandu.

Le système TROLL

TROLL (Time-shared Reactive On-Line Laboratory) a été développé au MIT d'abord puis par le NBER (National Bureau of Economic Research) qui assure encore aujourd'hui sa maintenance. Il peut être implanté sur tout ordinateur IBM fonctionnant sous le système d'exploitation VM. C'est donc un système tout à fait spécifique non seulement à un ordinateur, mais aussi à un système d'exploitation.

Ce logiciel est facile à utiliser et d'apprentissage rapide. Il est entièrement interactif et il intègre toutes les tâches économiques. Il possède une documentation et un système de messages d'erreurs trés élaborés. Il n'est pas possible à l'utilisateur d'enrichir le logiciel par l'adjonction de nouveaux algorithmes. Néammoins TROLL offre un macro-langage qui lui permet de construire ses propres programmes constitués de commandes TROLL et de macro instructions.

La syntaxe du langage de commande est simple. Les commandes sont en

format libre. Le logiciel possède une hiérarchie de commandes à trois niveaux. Les commandes de niveau 1 permettent d'exécuter les quatre tâches suivantes: édition de fichiers, estimation, simulation, transformation de données. Les commandes de niveau 2 permettent d'effectuer les diverses composantes des grandes tâches. Par exemple dans l'édition des données, il existe des commandes pour la correction, l'addition, l'impression de valeurs. Avec les commandes de niveau 3, on peut introduire des paramètres communs aux différentes tâches (par exemple, rechercher un fichier ou spécifier l'intervalle de temps sur lequel s'applique la tâche).

Conclusion

On constate qu'il n'existe pas de système sur le marché international répondant simultanément aux critères ci-dessous:

- adaptation aux besoins spécifiques de l'utilisateur;

- portabilité sur plusieurs types de matériels et, en particulier compatibilité avec les matériels informatiques francais;

- facilité d'utilisation pour un non-informaticien;

- structure souple du logiciel permettant le remplacement aisé des modules le composant par d'autres correspondant à des algorithmes plus récents;

- possibilité de traiter des modèles de grosse taille au contraire des logiciels que nous venons de décrire et qui eux ne prennent en charge que des modèles de taille petite ou moyenne.

3. Caractéristiques de MODULECO

Le logiciel MODULECO prend en charge toutes les étapes d'une étude économétrique. C'est un système intégré, portable et extensible, facile à mettre à jour en raison de sa structure modulaire. Son apprentissage est rapide et son utilisation facile, notamment grâce à la documentation et à l'aide interactive accessible à l'utilisateur à tous moments.

Sa portabilité résulte de sa séparation en deux parties: une couche portable écrite en langage PASCAL (environ 170 000 lignes) et une couche non portable. Pour installer MODULECO sur un nouveau site informatique, il faut réecrire cette couche non portable. Sur HB68, système MULTICS, cette partie est écrite en PL1 (environ 400 lignes). Sur IBM 3033, TSO, elle est écrite en assembleur. Sur VAX, système UNIX, elle est écrite en langage C.

Afin d'adapter le logiciel à des besoins spécifiques de certaines catégories d'utilisateurs, il est possible de définir de nouvelles commandes par combinaison de commandes existantes et d'ordres simples d'affichage ou de saisie au terminal, de répétitions ou de tests. Ces nouvelles commandes peuvent aussi être des versions simplifiées de commandes existantes. Le logiciel est conçu pour permettre d'ajouter aisément des commandes à la bibliothèque.

Enfin il faut remarquer que MODULECO est conçu pour traiter des modèles de grande taille.

4. Présentation d'une étude économétrique à l'aide de MODULECO

Pour illustrer ce que nous avons exposé jusqu'à maintenant, nous vous proposons un exemple d'une étude économétrique réalisée avec le logiciel MODULECO. Cette étude porte sur un petit modèle d'école, oscillateur (J.D

Laffay, Revue d'Economie Politique, 1972, pp 1135-1171).

```
MODULECO 01/12/84
MODULECO :

/* Analyse des séries: étude de la corrélation entre
   revenu, investissement et consommation
*/

corr (y,i,cons) (donneesosc) $50 $68;
                        NOM DES VARIABLES
                        .................

          VAR 1 : Y                              VAR 2 : I
          VAR 3 : CONS

                        MATRICE DE CORRELATION
                        ......................

                    VAR 1        VAR 2        VAR 3
     VAR  1       1.0000000    0.9833893    0.9975119
     VAR  2       0.9833893    1.0000000    0.9703570
     VAR  3       0.9975119    0.9703570    1.0000000

MODULECO :

/* Régression linéaire de la variable y en fonction
   de la variable cons et de la variable y considérée aux
   instants t-1 et t-2
*/

reglin (y,y<-1>,y<-2>,cons) (donneesosc);

                    REGRESSION LINEAIRE

     SEGMENT D'ORIGINE : SERIES.DONNEESOSC

     PERIODE DE REGRESSION : DE £1950
                             A £1968


      DEGRES DE LIBERTE    :  14
```

```
      NOMBRE D'OBSERVATIONS: 17

Y =  1.060E+00*Y<-1> + (-3.764E-02)*Y<-2> +  3.599E-02*CONS ;
     ..........................................................
     .  COEFF          ECART_TYPE       STAT          MOYENNE DE.
     .                                   T            LA SERIE .
     .                                                        .
     .  1.0603228      0.2886358      3.6735662      310.47156 .
     . -0.0376410      0.2995558     -0.1256560      295.26327 .
     .  0.0359913      0.1732331      0.2077625      224.83471 .
     ..........................................................


     R CARRE  :   0.9998872          SOMME DES CARRES
     R2AJUSTE :   0.9998710          DES RESIDUS :   217.31886
     SOMME DES                       MOYENNE DE LA VARIABLE
     RESIDUS  :  -0.1999124          EXPLIQUEE   :   326.16643
     ECART_                          F( 3,14)    :   41350.663
     TYPE     :   3.9398954          D_W         :   1.7442666

           ANALYSE DE LA VARIANCE
           ......................
     SOURCE DES     D.L.   SOMME DES   MOYENNE DES
     VARIATIONS             CARRES       CARRES

       REGRESSION     3    1925631.1     641877.05
       DEVIATION     14    217.31886    15.522775
       TOTAL         17    1925848.5

MODULECO :

/* Construction du modèle qui sera sauvegardé sous
   le nom lafay.oscillateur
*/

edm lafay.oscillateur %sauve;
NOUVEAU MODELE.
ENTREE.
MODELE OSCILLATEUR
```

```
/*AUTEUR : J.D.LAFFAY*/;

ENDO CONS,     /* CONSOMMATION DES MENAGES */
     TC, /* TAXES*/
     X, /* IMPORTATIONS */
     Y; /* REVENU */

EXO G, /* DEPENSES GOUVERNEMENTALES */
    E, /* EXPORTATIONS */
    INV; /* INVESTISSEMENTS */

COEFF A,B,PT,PTO,M,MO;

/*   EQUATIONS    */

CONS = A*(Y-TC)+B;
TC = PT*Y+PTO;
X = M*Y+MO;
Y = CONS+INV+G+E-X;
FIN OSCILLATEUR;

EDITION .
fin
MODULECO :

/* Estimation des coefficients de ce modèle à partir des
   valeurs des séries contenues dans le segment donneesosc.
   Construction du contexte d'estimation.
*/

estimer liseg=(donneesosc) ;
MODULECO :

/* La méthode d'estimation choisie est la méthode
   des moindres carrés ordinaires
*/

mco;
```

ESTIMATION DES COEFFICIENTS

SEGMENT D'ORIGINE : SERIES.DONNEESOSC

PERIODE D'ESTIMATION : DE £1950
A £1968
NOMBRE D'OBSERVATIONS: 19

EQUATION 1

DEGRES DE LIBERTE : 17

CONS = 7.723E-01*(Y-TC)+ 2.022E+01

COEFF	ECART_TYPE	STAT T	MOYENNE DE. LA SERIE
0.7722685	0.0142434	54.219292	253.73238
20.223103	3.7580181	5.3813214	1.0000000

R CARRE : 0.9942504
R2AJUSTE : 0.9939122
SOMME DES RESIDUS : 2.115E-14
ECART_TYPE : 4.4910819

SOMME DES CARRES DES RESIDUS : 342.88688
MOYENNE DE LA VARIABLE EXPLIQUEE : 216.17263
F(1,17) : 2939.7316
D_W : 0.6016195

ANALYSE DE LA VARIANCE

SOURCE DES VARIATIONS	D.L.	SOMME DES CARRES	MOYENNE DES CARRES
REGRESSION	1	59293.846	59293.846
DEVIATION	17	342.88688	20.169816
TOTAL	18	59636.733	

EQUATION 2

DEGRES DE LIBERTE : 17

TC = 1.744E-01*Y+ 4.596E+00

COEFF	ECART_TYPE	STAT T	MOYENNE DE. LA SERIE .
0.1744330	0.0059223	29.453495	312.91079 .
4.5964401	1.9243789	2.3885318	1.0000000 .

R CARRE :	0.9807803	SOMME DES CARRES DES RESIDUS :	86.900261
R2AJUSTE :	0.9796497	MOYENNE DE LA VARIABLE EXPLIQUEE :	59.178407
SOMME DES RESIDUS :	3.116E-15	F(1,17) :	867.50838
ECART_ TYPE :	2.2609246	D_W :	0.9103002

ANALYSE DE LA VARIANCE

SOURCE DES VARIATIONS	D.L.	SOMME DES CARRES	MOYENNE DES CARRES
REGRESSION	1	4434.5120	4434.5120
DEVIATION	17	86.900261	5.1117800
TOTAL	18	4521.4123	

EQUATION 3

DEGRES DE LIBERTE : 17

X = 1.975E-01*Y+(-1.595E+01)

COEFF	ECART_TYPE	STAT T	MOYENNE DE. LA SERIE .
0.1975339	0.0098747	20.003967	312.91079 .
-15.954700	3.2086642	-4.9723805	1.0000000 .

```
R CARRE  :    0.9592481          SOMME DES CARRES
R2AJUSTE :    0.9568510          DES RESIDUS :    241.59530
SOMME DES                        MOYENNE DE LA VARIABLE
RESIDUS  :    2.161E-15          EXPLIQUEE   :    45.855789
ECART_                           F( 1,17)    :    400.15871
TYPE     :    3.7698127          D_W         :    0.3995296

          ANALYSE DE LA VARIANCE
          ......................
SOURCE DES      D.L.    SOMME DES    MOYENNE DES
VARIATIONS              CARRES       CARRES

  REGRESSION      1     5686.8508    5686.8508
  DEVIATION      17     241.59530    14.211488
  TOTAL          18     5928.4461

EQUATION 4
**** ATTENTION : 5005
     PAS DE COEFFICIENTS A ESTIMER DANS L'EQUATION
MODULECO :

/* Fin du contexte d'estimation
*/

finest;
MODULECO :

/* Transformation des segments de séries en un segment
   transposé, c'est à dire un segment où les donnés
   sont rangées période par période au lieu de l'être
   série par série
*/

trp liseg=(donneesosc) segst=donneest;
MODULECO :

/* Réordonnancement du modèle pour minimiser le
   nombre de variables de bouclages et ainsi diminuer le
```

```
   temps de simulation
*/

ordo;
ORDONNANCEMENT DU BLOC OSCILLATEUR (4 VARIABLES)
ALGORITHME TERMINE.  TEMPS CPU=0"29
MODULECO :

/* Simulation du modèle
*/

sml;
SIMULATION DE LA VARIANTE BASE
SIMULATION TERMINEE.  TEMPS CPU=3"19
MODULECO :

/* Impression des résultats de la simulation
   pour les variables de bouclage
*/

ist base liser=liboucl();

                    SEGMENT TRANSPOSE LAFAY.OSCILLATEUR.TRANSP.BASE

     PERIODICITE : ANNUEL

DATE                 Y

1950          205.316368
1951          213.288532
1952          220.575743
1953          219.816489
1954          230.213384
1955          241.716226
1956          256.974527
1957          271.920466
1958          276.143655
1959          295.069769
1960          324.711362
1961          335.448817
```

```
1962        351.931979
1963        365.799006
1964        395.290240
1965        408.286452
1966        438.317419
1967        455.593562
1968        473.549468

MODULECO :

/* Modification dela première équation du modèle
*/

edm lafay.oscillateur %modif;
MODIFICATION D'EQUATIONS
EDITION.
p.CONS =.
1: CONS = A*(Y-TC)+B;
c.Y-TC).Y-TC)<-1>.
1: CONS = A*(Y-TC)<-1>+B;
fin
MODULECO :

/* Ajout d'une équation
*/

edm ;

AJOUT D'EQUATIONS
ENTREE.
endo inv;
inv=(c*y)<-1..-2>+c;

EDITION.
fin
MODULECO :

/* Sauvegarde du modèle
*/
```

```
stomod;
MODULECO :

/* Puisque le modèle a été modifié, il
   faut refaire une estimation des coefficients
*/

est liseg=(donneesosc) ;
MODULECO :
mco %nosort;
MODULECO :
finest;
MODULECO :

/* Impression des coefficients
*/

isc;

          SEGMENT DE COEFFICIENTS LAFAY.OSCILLATEUR.ESTIM.RESEST.COEFF

    COEFFICIENTS NOMMES

 M0 :      -18.041239 M :       0.203004 PT0 :       3.255544
 PT :        0.177958 B :      20.207485 A :       0.813032

    COEFFICIENTS NON NOMMES

EQUATION   5
  C1  :       0.529556  C2  :      -0.319062  C3  :     -29.721180

MODULECO :

/* Réordonnancement du modèle
*/

ordo;

ORDONNANCEMENT DU BLOC OSCILLATEUR (5 VARIABLES)
ALGORITHME TERMINE.  TEMPS CPU=0"14
```

```
MODULECO :

/* Consultation du modèle: les principaux
   renseignements le concernant sont affichés
*/

consmod;

MODELE          OSCILLATEUR
CALENDRIER      ANNUEL
BLOC NON NORMALISE
PAS DE CODE FORTRAN EXISTANT
NOMBRE D'EQUATIONS     :   5
NOMBRE D'ENDOGENES     :   5
TAILLE DU PROLOGUE     :   2
TAILLE DU COEUR        :   1
TAILLE DE L'EPILOGUE   :   1
VARIABLES DE BOUCLAGE  :   1

NOMBRE D'EXOGENES      :   2
DECALAGE MAXIMUM POUR LE MODELE :    2

MODULECO :

/* On recommence la simulation avec cette nouvelle forme
   de modèle
*/

sml resul=res1;
SIMULATION DE LA VARIANTE BASE
SIMULATION TERMINEE.  TEMPS CPU=2"46
MODULECO :

/* Comparaison des résultats des deux simulations
   pour les variables de bouclages
*/

cst base res1 liser=liboucl();

                    COMPARAISON DES SEGMENTS TRANSPOSES
```

LAFAY.OSCILLATEUR.TRANSP.BASE
LAFAY.OSCILLATEUR.TRANSP.RES1

ECARTS RELATIFS EN POURCENTAGE

PERIODICITE : ANNUEL

DATE	Y
1952	0.152450
1953	-6.523340
1954	-6.980157
1955	-6.448832
1956	-4.217300
1957	-2.467197
1958	-3.613384
1959	-0.033049
1960	4.920853
1961	2.894321
1962	2.641106
1963	1.429653
1964	3.872376
1965	1.131841
1966	2.141127
1967	0.008958
1968	-1.929199

MODULECO :

/* Fin de la session
*/

logout;
LE MODELE COURANT : LAFAY.OSCILLATEUR N'A PAS ETE SAUVEGARDE
VOULEZ-VOUS LE SAUVEGARDER ?
oui

CHAPITRE III

LE LANGAGE DE COMMANDE ET SON ENVIRONNEMENT

1. Le langage de commandes ou langage MODULECO

2. Rôle des commandes et avantages

3. Description détaillée d'une commande

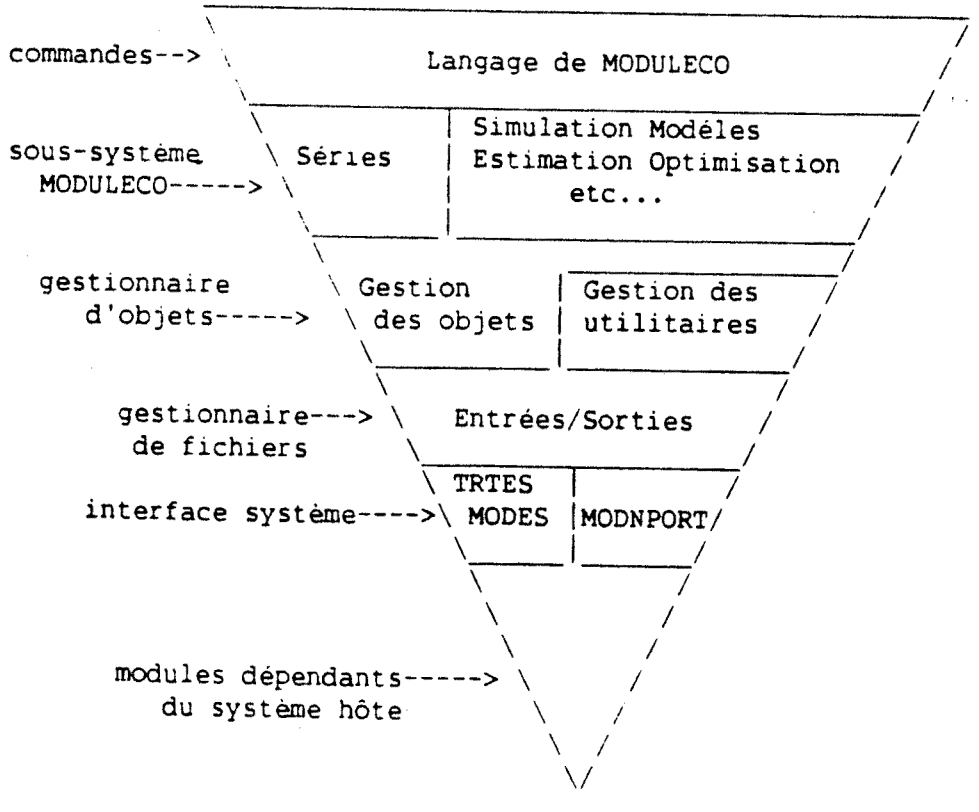
Jusqu'à maintenant nous avons présenté le langage de commandes comme un langage d'interaction entre le système et l'utilisateur, c'est à dire comme un outil permettant à l'utilisateur de mettre en oeuvre le logiciel et de déclencher, grâce aux instructions de ce langage ou commandes, toute une séquence de fonctions élémentaires. Ceci n'est qu'un des rôles de ce langage qui en somme est beaucoup plus qu'un simple langage de commandes. C'est pourquoi, nous avions pensé à le renommer langage MODULECO. En fait, on utilise maintenant indifféremment les deux termes.

1. Le langage de commandes ou langage MODULECO

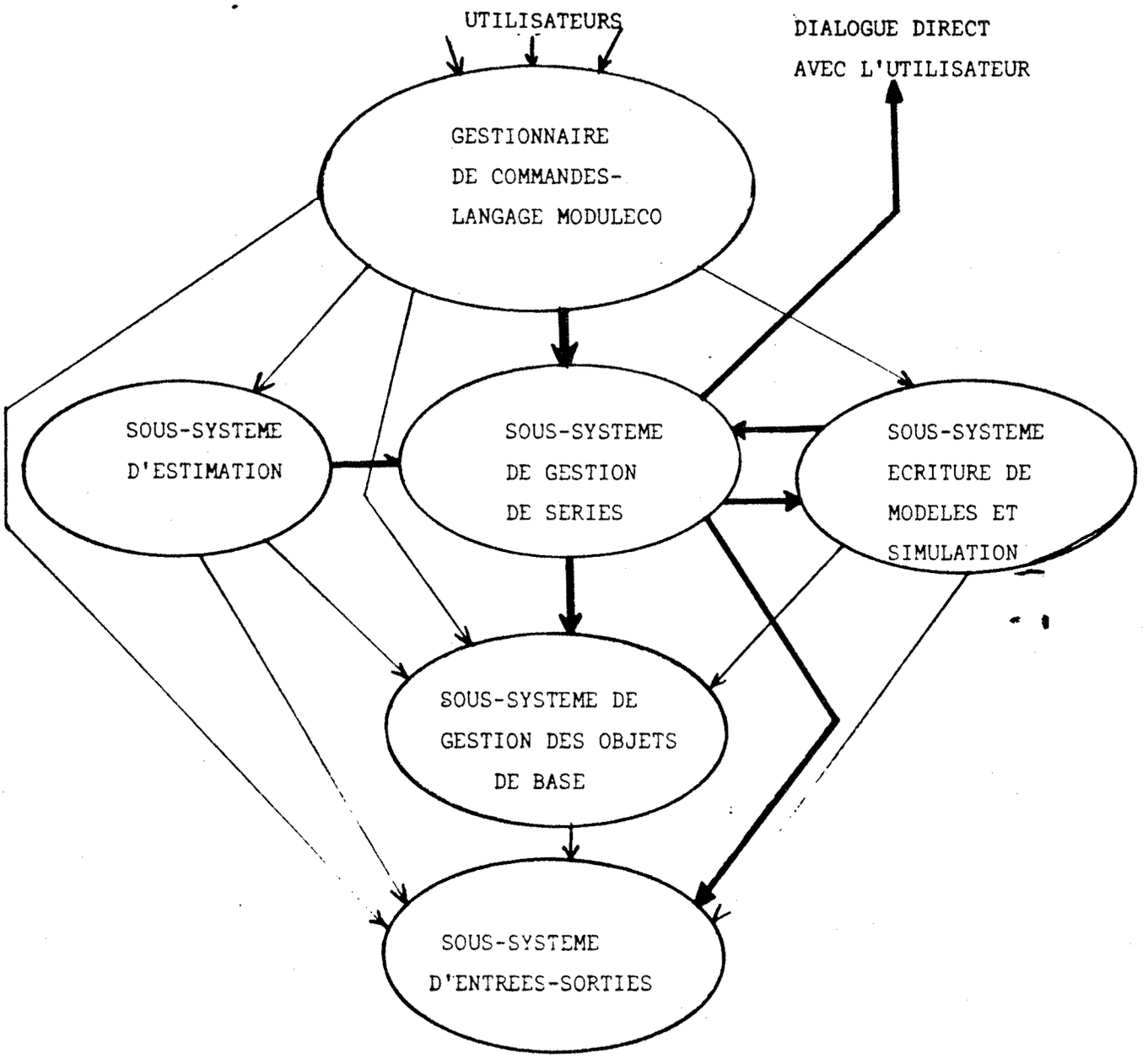
Voici une description des différentes fonctions de ce langage.

C'est le lien entre les différents sous-systèmes qui composent le logiciel

Le schéma ci-dessous montre le découpage en couches de MODULECO.



Nous nous intéressons uniquement à la couche sous-systèmes de MODULECO que nous vous détaillons ci-dessous.



L'utilisateur n'accède qu'au niveau "commandes" qui lui, appelle les sous-systèmes concernés. Grâce aux paramètres des commandes et spécialement à ceux de types spécifiques aux objets MODULECO, les résultats obtenus dans un composant de MODULECO peuvent être exploités dans un autre. Le langage de commandes sert ici d'interface entre les différents sous-systèmes qui composent le logiciel.

C'est un langage de programmation

Il permet à l'utilisateur d'enrichir le langage par ajout de nouvelles commandes et d'adapter le logiciel à des besoins spécifiques. Les programmes de commandes peuvent être des procédures ou des fonctions et regrouper des appels à d'autres fonctions ou procédures existantes en enchaînant leurs exécutions. C'est donc essentiellement un langage algorithmique évolué travaillant sur des types de données spécifiques. Cet aspect du langage est détaillé dans le chapitre suivant, premier paragraphe.

C'est un langage de base

Il offre un ensemble assez complet de fonctions de base. Actuellement le logiciel comprend 110 commandes standard; tous les domaines nécessaires à une étude économétrique sont couverts.

Il offre de plus des moyens puissants et nombreux pour assembler et combiner ces fonctions de base afin de créer de nouvelles commandes spécifiques à une application donnée. Nous verrons plus loin comment, pour un modèle donné, on peut facilement rajouter une nouvelle méthode d'estimation des coefficients des équations par combinaison de commandes existantes, ceci dans le cas où les méthodes existantes se révèlent inadaptées au problème donné.

2. Rôle des commandes et avantages

Une étude économétrique comportent plusieurs étapes:

-création, modification et visualisation des données économiques,

-spécification et édition de modèles,

-estimation de la valeur des coefficients apparaissant dans les équations,

-simulation des modèles obtenus,

-conservation et présentation des résultats.

Ces différentes étapes se décomposent elles mêmes en tâches élémentaires. A chacune d'elles est associée une commande MODULECO pour permettre l'appel de l'algorithme concerné. Ce programme décrit l'enchaînement des actions élémentaires nécessaires à l'exécution du travail demandé par l'utilisateur. Les paramètres de la commande correspondent aux informations utiles à l'exécution de l'algorithme.

Tous les travaux "non économiques" sont ainsi évités à l'utilisateur. Il n'a pas à se préoccuper de gestion de fichiers, de mémorisation de résultats ni d'accés aux données ou de format d'édition des résultats. Tout ceci est pris en charge par la commande. De plus chaque commande est conçue pour prendre le maximum d'initiatives et demander le minimum d'information a l'utilisateur.

Une aide interactive est à tout moment à la disposition de ce dernier. Pour chaque commande, ses fonctionnalités sont spécifiées dans une documentation associée. A chaque paramètre est attaché une documentation relative à son rôle. De plus on peut connaitre à chaque instant le nom et le type du prochain paramètre dont on doit fournir la valeur.

La plupart des paramètres de la commande sont facultatifs et possèdent une valeur par défaut. Dans presque tous les cas, il n'y a donc pas besoin de les spécifier. Au premier appel de la commande on utilise les valeurs par défaut définies à la création de la commande, ou valeurs par défaut système. Sur ce

premier appel, les valeurs par défaut seront écrasées par les valeurs entrées par l'utilisateur ou valeurs par défaut utilisateur. Ainsi, au cours d'une session, on n'indique que les valeurs des paramètres qu'on désire modifier.

Au cours d'une session, un contexte d'exécution est défini peu à peu. Des variables "courantes" sont affectées, par exemple le modèle de travail ou le segment de série de référence. Le langage MODULECO met aussi à la disposition de l'utilisateur des variables "globales" qu'il définit lui même qu'il affecte et qu'il peut utiliser dans les appels de commande. Ces variables lui sont personnelles.

L'appel d'une commande est analysé ligne par ligne. Lorsque des erreurs sont détectées, elles sont signalées et expliquées par un message clair. Il est alors possible de corriger l'appel erroné sous l'éditeur MODULECO et de relancer la commande. Certaines commandes comportant une quarantaine de paramètres il était indispensable de fournir un mécanisme de ce genre.

Trois possibilités sont offertes pour fournir les paramètres de la commande:

-entrée des paramètres dans l'ordre spécifié à la création de la commande; la première valeur est affectée au premier paramètre, la ième au ième paramètre. Lorsqu'on rencontre deux virgules consécutives, le paramètre qui occupe cette position recoit sa valeur par défaut s'il en a une ou n'est pas affecté sinon.

Exemple:

sml ,,(donneesosc),,,,,resec simec %impvari;

-entrée des paramètres dans un ordre quelconque; dans ce cas l'ordre des paramètres est quelconque et on donne le nom du paramètre suivi du signe égal suivi de sa valeur.

Exemple:

sml resul=simec lientr=(donneesosc) ecart=resec %impvari;

-entrée conversationnelle des paramètres; l'utilisateur se laisse guider par le

logiciel qui fournit le nom, le type et la documentation du paramètre attendu. Pour cela il suffit d'envoyer un retour chariot (RC) après le nom de la commande et après la valeur de chaque paramètre. Pour indiquer qu'un paramètre n'est pas affecté, on tape une virgule. L'utilisateur peut utiliser cette facilité pour chaque paramètre ou seulement pour certains. Les trois possibilités peuvent être mélangées.

Exemple:

```
sml lientr=(donneesosc)
< DD            DATE  >
DATE DE DEBUT DE SIMULATION
,
< DF            DATE  >
DATE DE FIN DE SIMULATION
,
< PARAM         SYMBOLIQUE IDENT  >
SEGMENT DES PARAMETRES
,
< COEFF         SYMBOLIQUE IDENT  >
SEGMENT DES COEFFICIENTS
,
< ECART         SYMBOLIQUE IDENT  >
SEGMENT DES VARIABLES D'ECART
resec
< RESUL         SYMBOLIQUE IDENT  >
%SEGMENT TRANSPOSE DES RESULTATS
simec
< RELAX         SYMBOLIQUE IDENT  >
SEGMENT COEFFICIENTS RELAXATION GAUSS-SEIDEL
%impvari;
```

Les commandes sont organisées en arborescence pour gérer des paramètres communs et assurer une cohérence contextuelle dans l'appel des tâches. Par exemple, les différents algorithmes d'estimation ont besoin d'un ensemble de renseignements communs tels que le nom du modèle et le nom des segments de série. On distingue une commande mère ESTIMER qui précise un

contexte d'estimation en affectant une valeur aux paramètres communs à toutes les méthodes d'estimation. Un ensemble de commandes, filles d'ESTIMER, est ensuite défini. Elles ne peuvent s'exécuter que si la commande mère a déja été appelée, c'est à dire si un contexte d'exécution a été défini. Dans ce cas précis, les commandes filles correspondent à l'ensemble des méthodes d'estimation proposées par le logiciel.

Le logiciel est livré avec un ensemble de commandes de base (110 actuellemnt) qui permettent de mener à bien une étude économétrique. Un générateur de rapport permet de présenter les conclusions de cette étude. Un outil graphique trés puissant a été rajouté dernièrement et permet de visualiser des séries en fonction du temps ou les unes par rapport aux autres.

3. Description détaillée d'une commande

Une commande est identifiée par un nom et une abréviation facultative; ses fonctionnalités sont spécifiées dans une documentation associée, obligatoire. Le principe suivant a été adopté: les premières lignes de la documentation explique le rôle de la commande et les résultats qu'elle fournit; ensuite la signification et la valeur par défaut de chaque option est indiquée. Un entête suivant la syntaxe du langage MODULECO liste les paramètres et les options. Un paramètre est décrit par son nom, son type et une documentation relative à sa fonction. Il est obligatoire ou facultatif pour l'exécution de la commande. Nous avons parlé plus haut des valeurs par défaut. Les options permettent de préciser un traitement bien particulier; elles ont toujours une valeur par défaut: par exemple l'option NONEXO de certaines commandes signifie "Ne pas appliquer ce traitement aux variables exogènes du modèle.". Si cette option n'est pas positionnée, le traitement est appliqué à toutes les variables du modèle, qu'elles soient exogènes ou endogènes.

Des paramètres (ou des options) sont dits mutuellement exclusifs si la présence de l'un à l'appel de la commande interdit la présence du ou des autres

paramètres (ou options) avec lesquels il est exclusif. Un groupe de paramètres exclusifs est:

soit facultatif et dans ce cas l'utilisateur peut trés bien n'en fournir aucun;

soit obligatoire et dans ce cas il faut nécessairement en affecter un.

Par exemple, dans la commande TRANSPOSER, on fournit soit le paramètres LISER (liste des séries à transposer), soit le paramètre LISAUF (liste des séries à ne pas transposer), soit ni l'un ni l'autre (transposition de toutes les séries du modèle).

On trouve également ce caractère d'exclusivité entre paramètres et options. Dans ce cas la donnée du paramètre et de l'option correspondante sont contradictoires.

Les paramètres peuvent désigner différentes catégories de données: valeurs numériques, dates, mais aussi objets de MODULECO (modèles, segments ou annuaires). On dira que les paramètres peuvent être de type différent. Les valeurs attribuées à ces paramètres doivent respecter une syntaxe particulière, propre au type du paramètre. On distingue dans MODULECO cinq types élémentaires: entier, réel, logique, date, chaîne de caractères et symbolique. A ceux-ci viennent s'ajouter des types structurés: les listes, suites d'éléments de type élémentaire.

On ne s'étendra pas sur les types entier, réel et logique. Pour les paramètres de *type date*, la syntaxe varie selon le calendrier de référence. Dans tous les cas une date commence par le caractère réservé dollar ($) ou dièse (#).

Exemple: #1970, $70 ces valeurs représentent l'année 70

$iv/1970, #iv/70 4ème trimestre de l'année 70

$1/1970, #janv/70 Janvier 1970

Une valeur de type *chaîne* est représentée par une chaîne de caractères incluse entre deux simples quotes (').

Exemple: 'C''est une chaîne.'

Les paramètres de *type symbolique* permettent de désigner des objets de MODULECO: modèle, segments de données, paramètres, coefficients, segments transposés, séries, équations. Le type symbolique admet des sous-types dont voici une liste non exhaustive:

-*sous-type IDENT* (abréviation d'identificateur): c'est une suite de seize lettres ou chiffres au plus, le premier caractère étant obligatoirement une lettre.

Exemple: ALLEMAGNE, france

-*sous-type IDSER* (abréviation d'identificateur de séries): c'est le nom d'une série chronologique.

Exemple: import (france, allemagne), cette série fait référence aux importations de la France en provenance de l'Allemagne

import (*, allemagne), cette série fait référence aux importations de tous les pays en provenance de l'Allemagne.

-*sous-type IDPT* (abréviation d'identificateur pointé): identificateur ou suite d'identificateurs séparés par des points. Le nom d'un modèle est un identificateur pointé; le premier désigne le nom du projet, le deuxième le nom du modèle.

Exemple: dms.minidms

-*sous-type IDEQ* (abréviation d'identificateurs d'équations): c'est le nom d'une équation

Exemple: EQ14

BL1>14, c'est la quatorzième équation du premier bloc

-sous-type DOC (abréviation de documentation): c'est une chaîne de caractères entre accolades ou entre (/*) et (*/).

-sous-type TXTL3E (texte dans le langage de modélisation L3E): suite d'une ou plusieurs équations séparées par un point-virgule (;). Le texte en L3E doit être entre guillemets (").

Exemple: "cons=rev*propcons;"

Les types structurés: on distingue deux types de listes, les listes homogènes et les listes hétérogènes.

-les listes homogènes:

Une liste homogène est une suite de valeurs d'un même type, donnée entre parenthèses. Chaque valeur est séparée de la suivante par une virgule. La liste peut avoir jusqu'à trois dimensions. Dans ce cas un élément de la liste est lui-même une liste.

Exemple: ($1970,#80,$81) liste de dates

(france,allemagne,italie) liste de symbolique ident

((1.2,3.4,-5.6),(-0.75,1.0e4,26.0)) liste de liste de réels

-les listes hétérogènes:

Une liste hétérogène est une suite d'éléments de type différents.

Exemple: ((vrai, 1, dms), (faux, 3, lafay)

Un élément de cette liste est composée de trois champs:

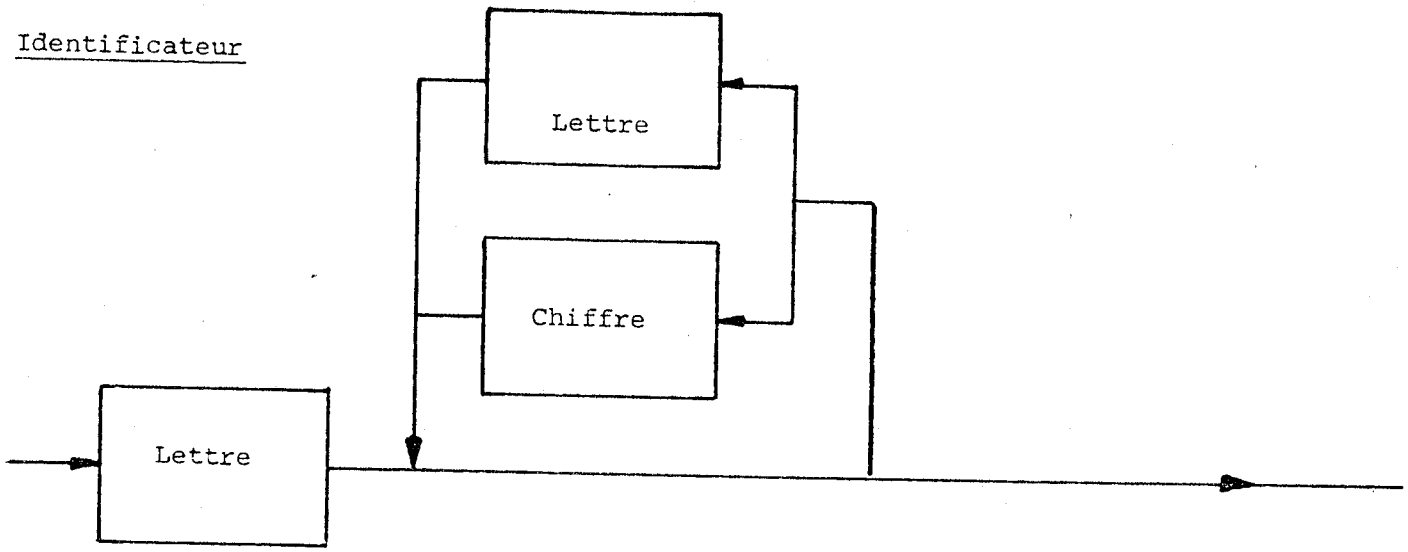
-un champ de type logique,

-un champ de type entier,

-un champ de type symbolique ident.

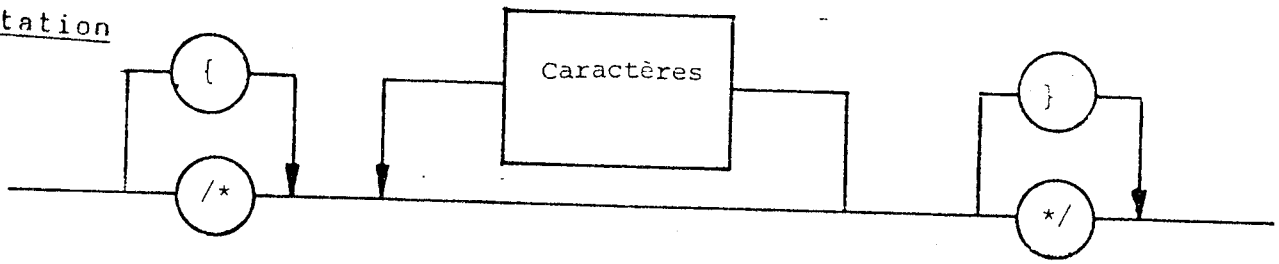
Voici les chartes syntaxiques correspondantes.

Identificateur



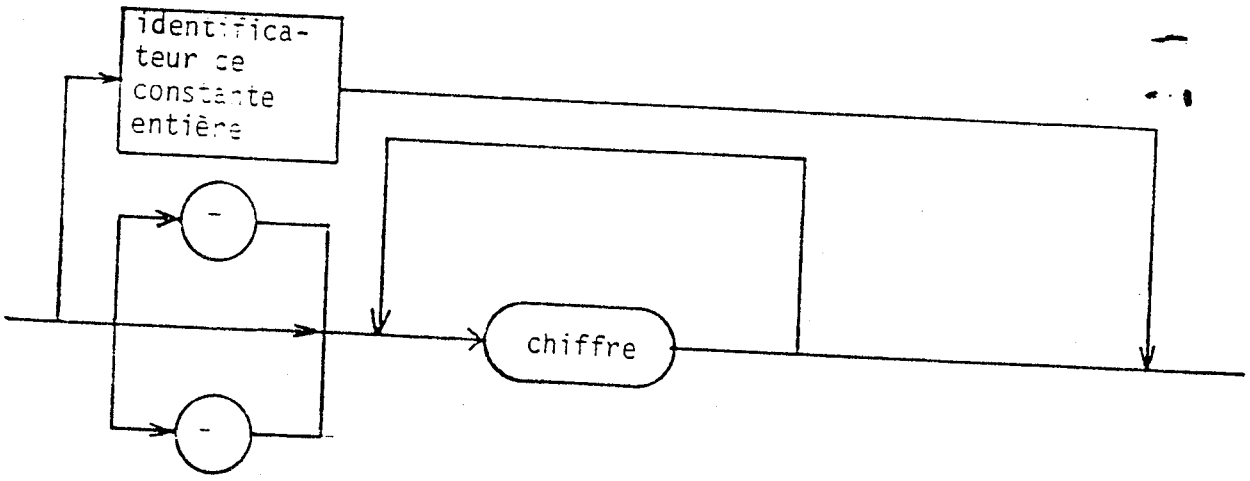
Charte 1

ocumentation



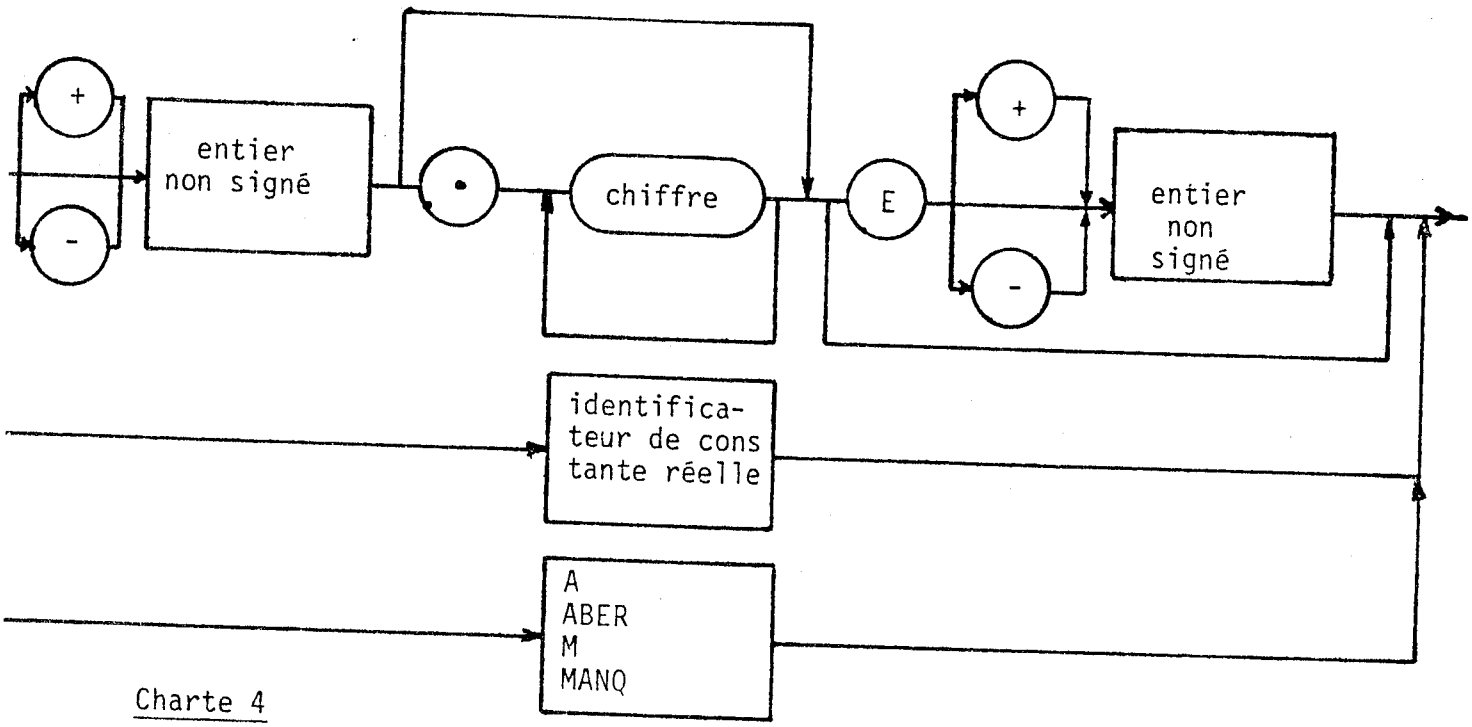
Charte 2

constante entière



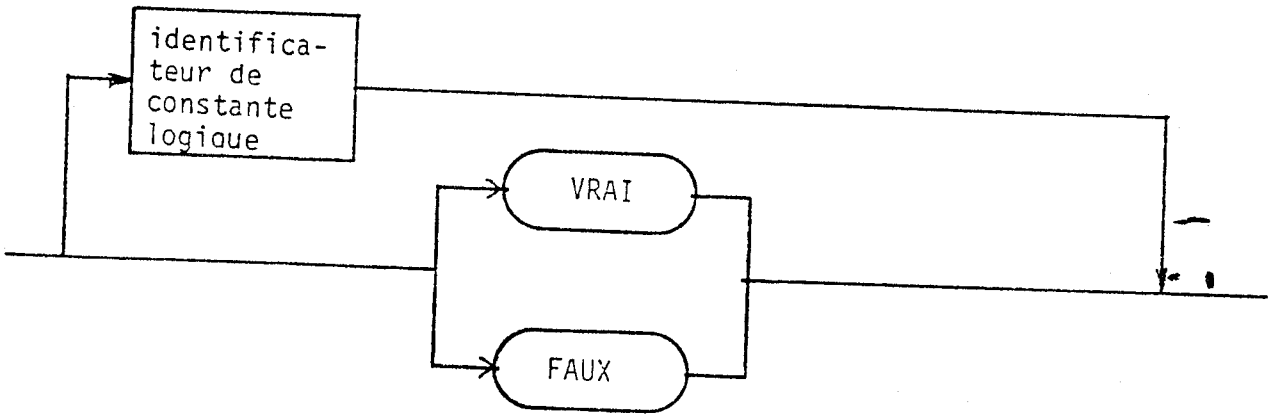
Charte 3

Constante réelle



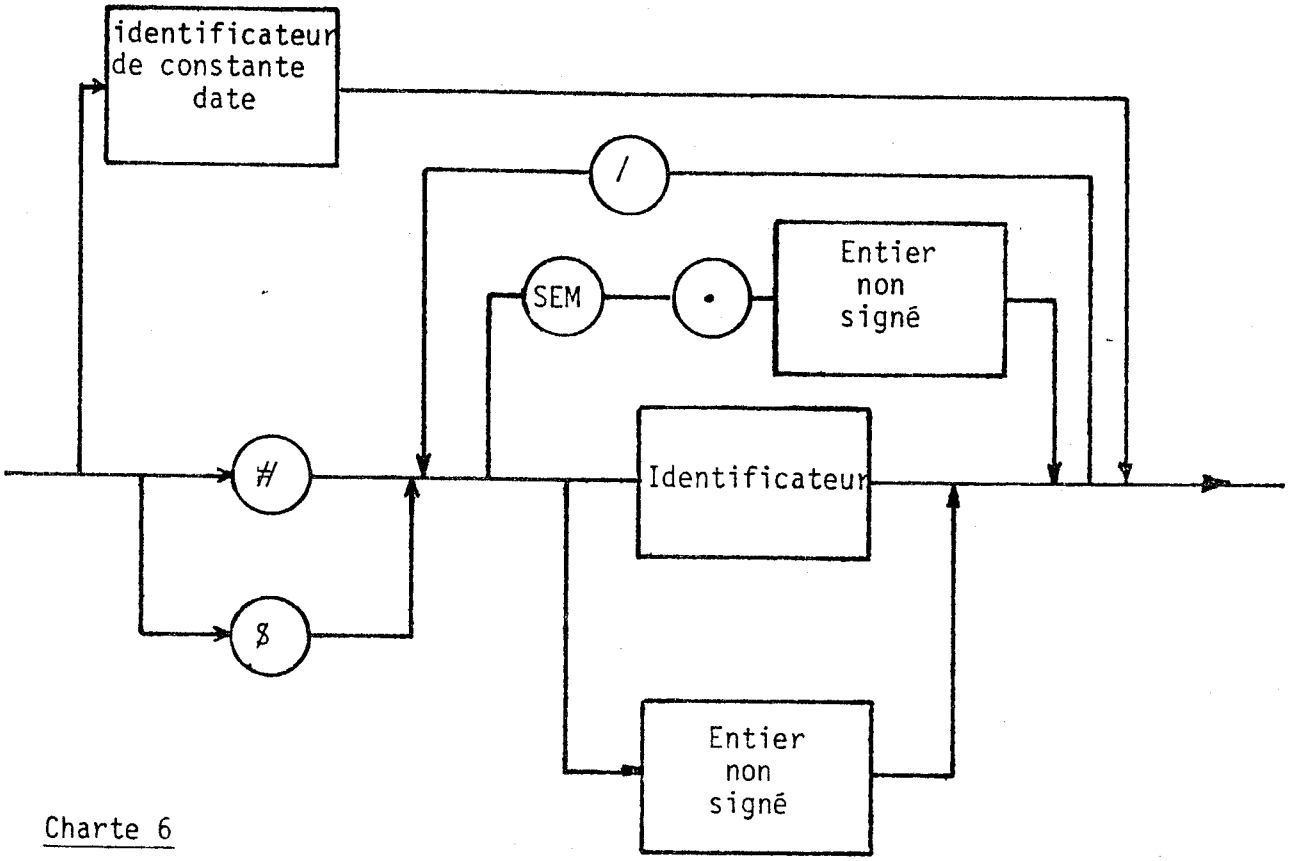
Charte 4

onstante
ogique



harte 5

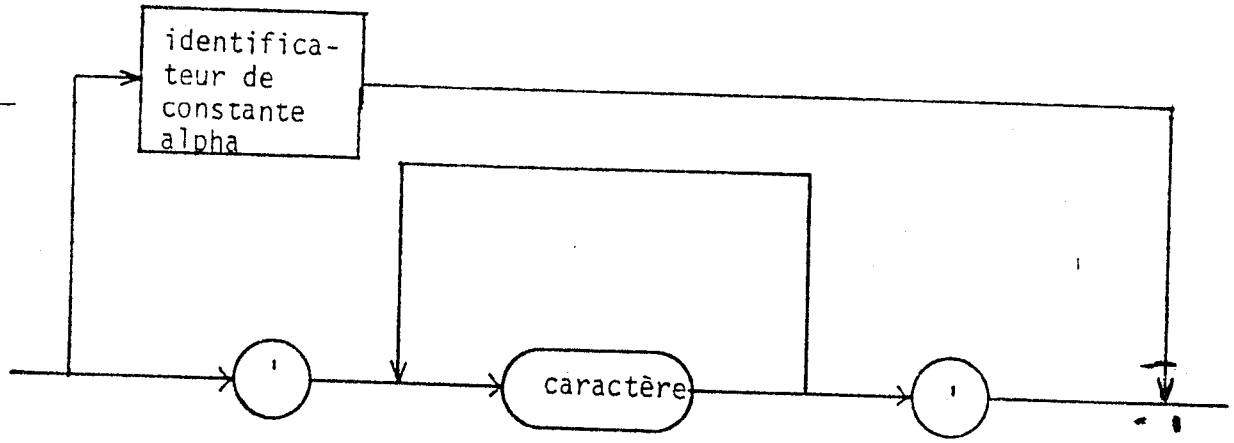
Les dates



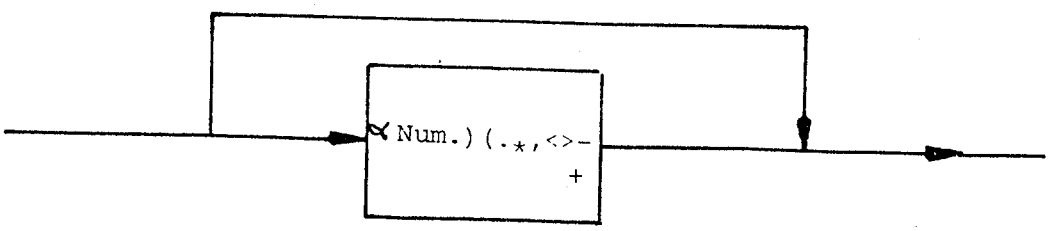
Charte 6

Chaîne de caractères

constante

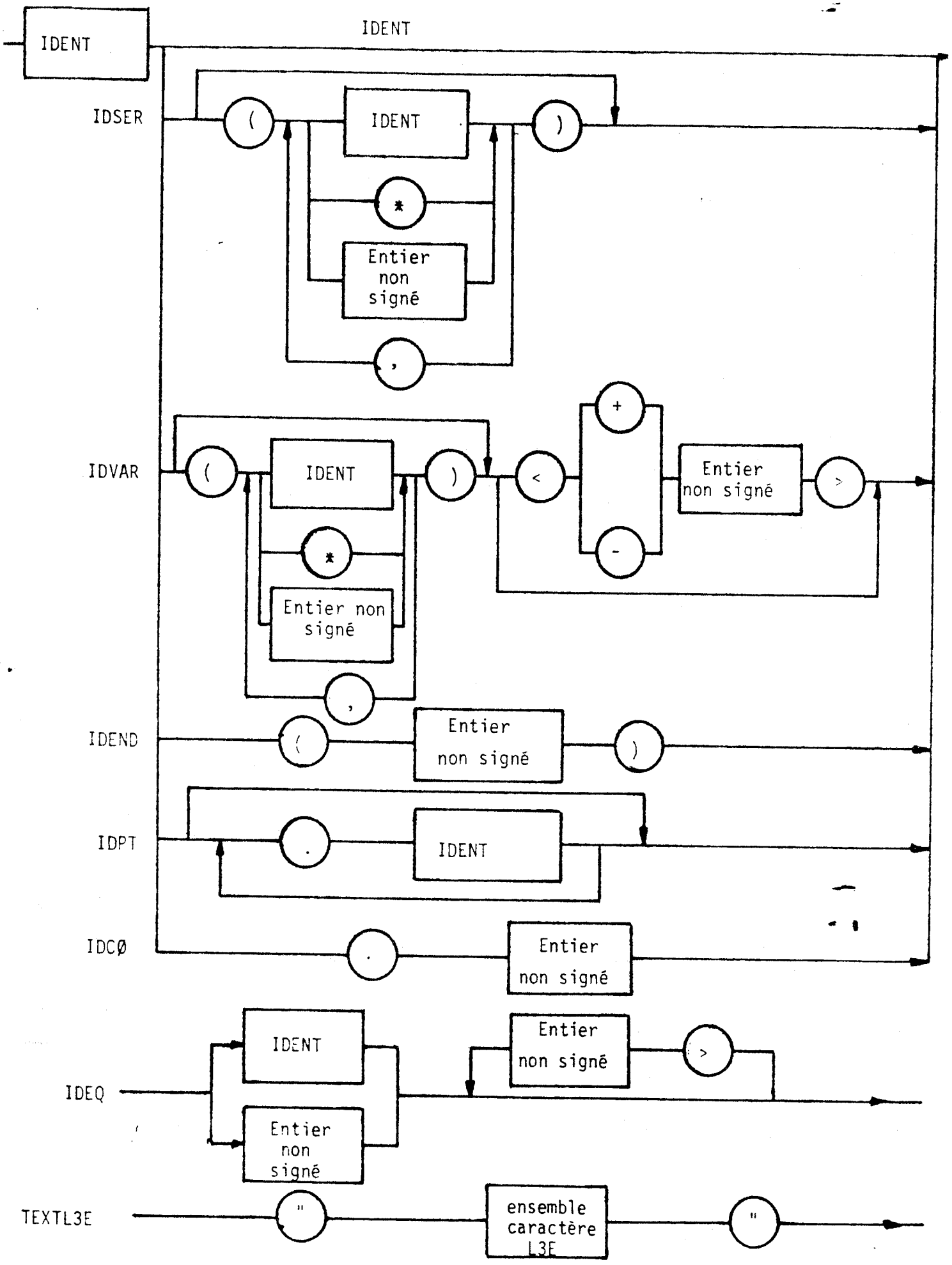


Charte 7



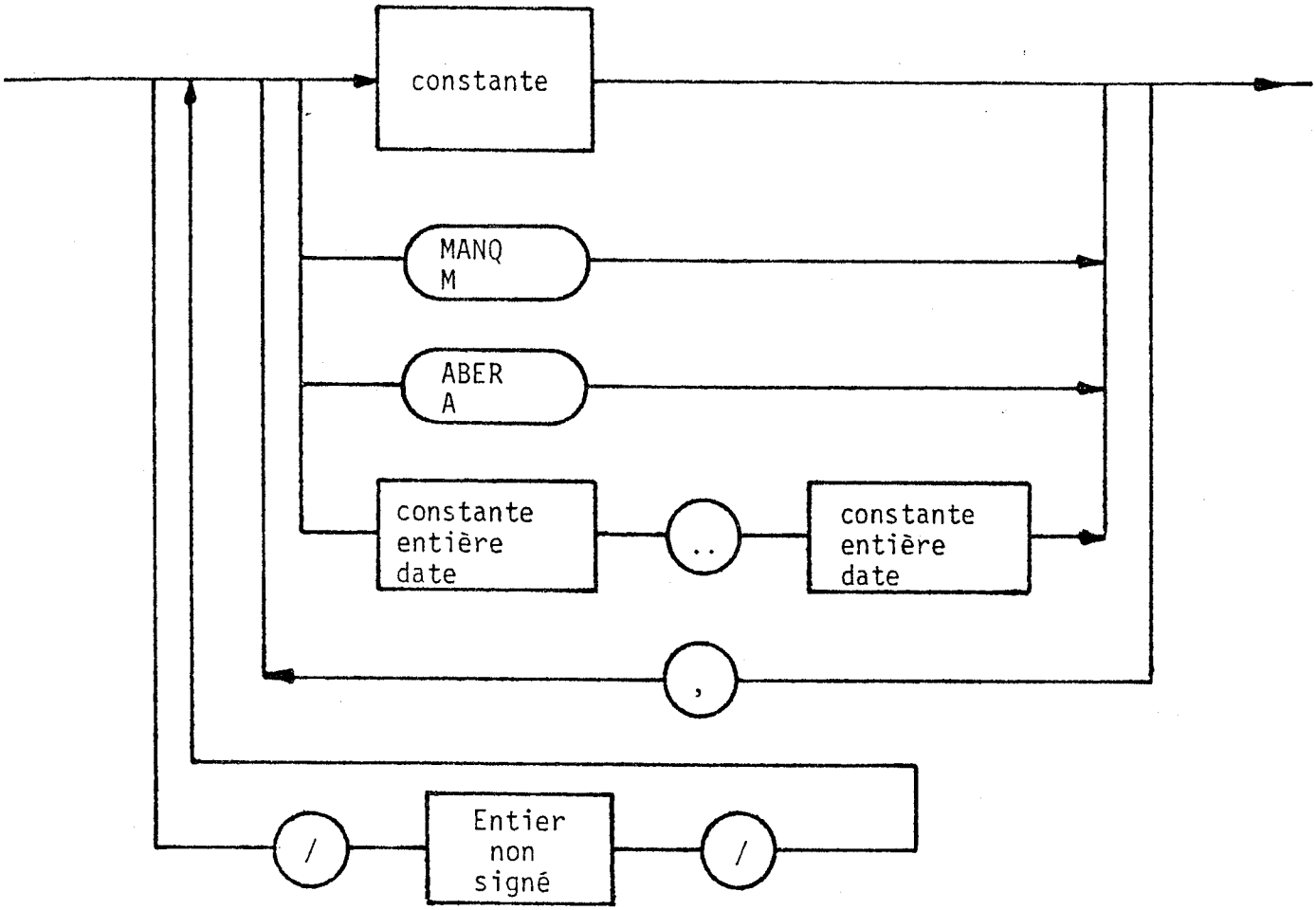
Symbolique Charte 8

Valeur de variable symbolique

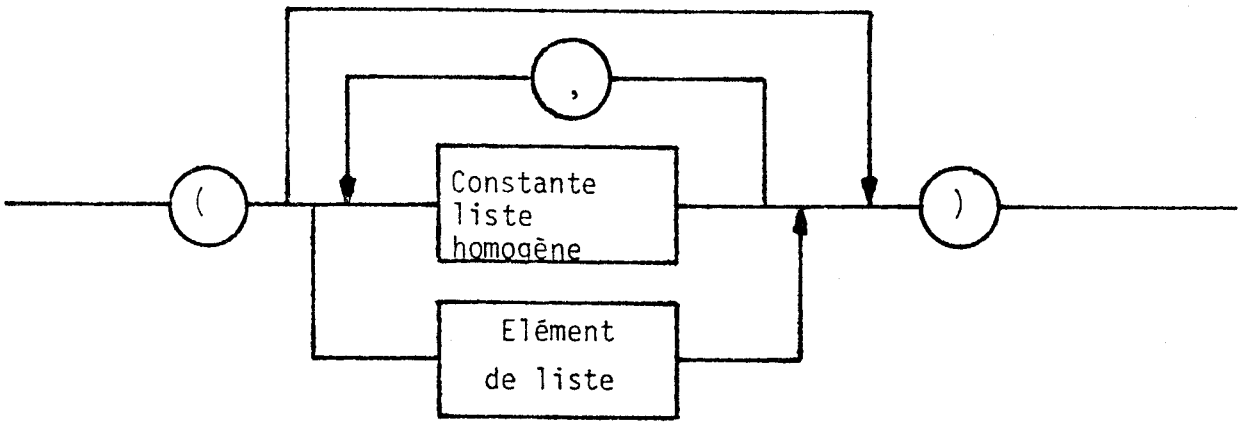


Carte 11

Elément de liste

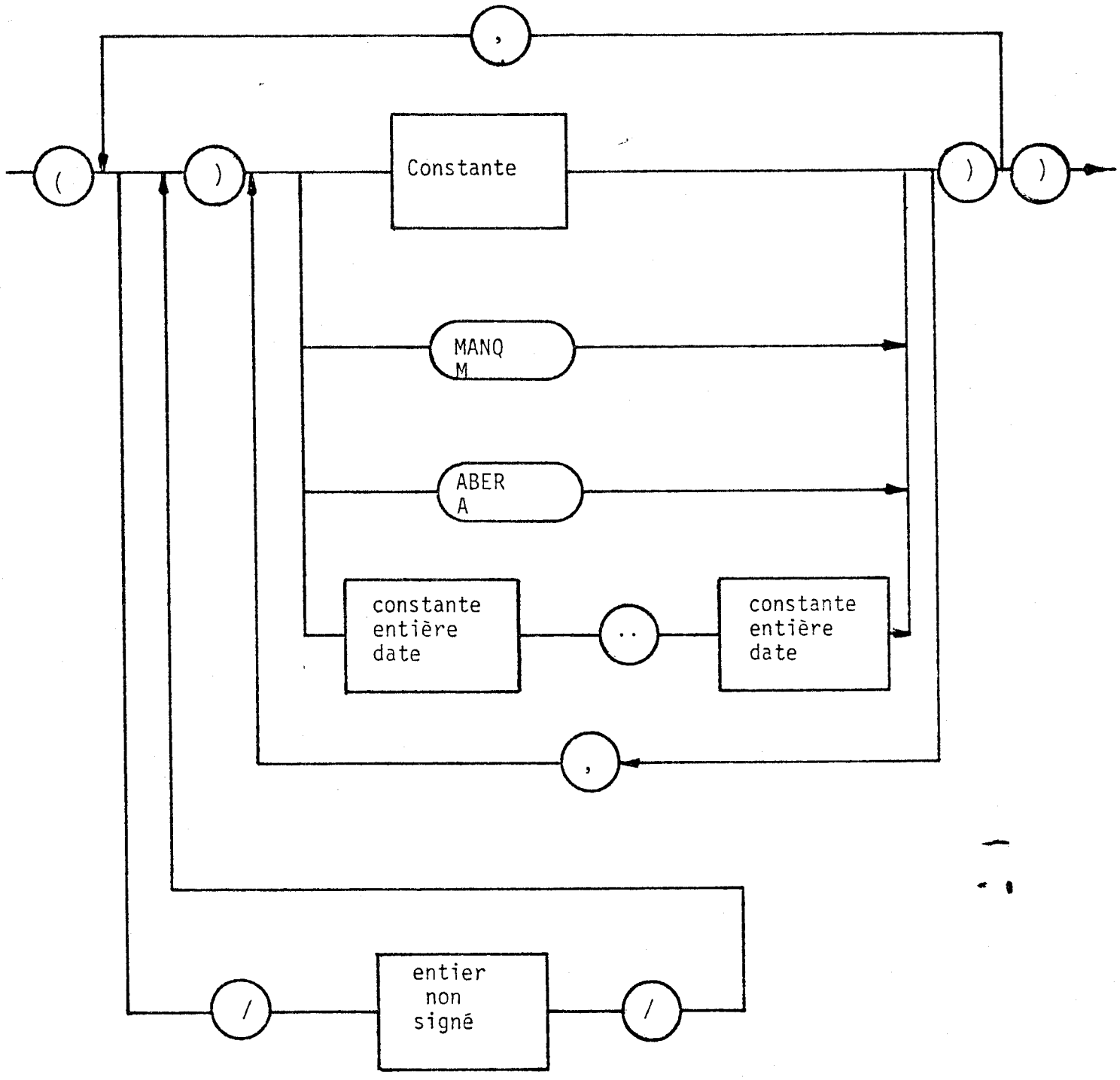


Constante liste homogène



Charte 12

Constante liste hétérogène



charte 14

Pour illustrer notre propos, voici la commande SIMULER. Nous donnons une présentation de cette commande telle qu'elle serait donnée à l'utilisateur qui en ferait la demande sous le système MODULECO.

```
SIMULER          SML
SIMULATION
OPTIONS IMPVARI : IMPRESSION DU CONTEXTE DE SIMULATION;
     NOSTOCK : LE CONTEXTE DE SIMULATION N'EST PAS SAUVEGARDE;
     VERIF   : VERIFICATION DE L'ALIMENTATION DES VARIABLES
< MOD            SYMBOLIQUE IDPT  >
NOM DU MODELE
< LIBLOC         LISTE HOMOGENE  SYMBOLIQUE IDENT  >
ENCHAINEMENT DES BLOCS
< LIENTR         LISTE HOMOGENE  SYMBOLIQUE IDENT  >
LISTE DES SEGMENTS TRANSPOSES D'ENTREE
< DD             DATE  >
DATE DE DEBUT DE SIMULATION
< DF             DATE  >
DATE DE FIN DE SIMULATION
< PARAM          SYMBOLIQUE IDENT  >
SEGMENT DES PARAMETRES
< COEFF          SYMBOLIQUE IDENT  >
SEGMENT DES COEFFICIENTS
< ECART          SYMBOLIQUE IDENT  >
SEGMENT DES VARIABLES D'ECART
< RESUL          SYMBOLIQUE IDENT  >
SEGMENT TRANSPOSE DES RESULTATS
< RELAX          SYMBOLIQUE IDENT  >
SEGMENT COEFFICIENTS RELAXATION GAUSS-SEIDEL
< JAC            SYMBOLIQUE IDENT  >
SEGMENT DES JACOBIENS DE NEWTON
< FREQ           ENTIER  >
FREQUENCE D'IMPRESSION DES RESULTATS INTERMEDIAIRES
< METH           SYMBOLIQUE IDENT  >
NOM DE LA METHODE DE RESOLUTION
< TEST           REEL  >
SEUIL DU TEST DE CONVERGENCE
```

< MAXIT ENTIER >
NOMBRE MAXIMUM D'ITERATIONS
< NCHIF ENTIER >
NOMBRE DE CHIFFRES SIGNIFICATIFS EN IMPRESSION
< CALJ ENTIER >
FREQUENCE DE CALCUL DU JACOBIEN POUR NEWTON
< LIIMPVAR LISTE HOMOGENE SYMBOLIQUE IDENT >
LISTE DES VARIABLES A IMPRIMER
< LIMODIF LISTE HOMOGENE SYMBOLIQUE IDENT >
LISTE DES VARIABLES A MODIFIER EN CONVERSATIONNEL
< LIRESUL LISTE HOMOGENE SYMBOLIQUE IDENT >
LISTE DES VARIABLES A STOCKER DANS LE SEGM.RESULTAT
< LIOPT LISTE HOMOGENE SYMBOLIQUE IDENT >
LISTE DES OPTIONS DE LA COMMANDE DE SIMULATION
A PRENDRE PARMI LES SUIVANTES:
INTERP,VARINIV,IMPMESS,IMPHISTO,IMPBOUC,
IMPJAC,TRACE,INITRESUL,AMORC,STATIQUE
< VARI SYMBOLIQUE IDENT >
NOM DE LA VARIANTE STOCKEE
OPTIONS
IMPVARI NOSTOCK VERIF

Les vingt deux paramètres de cette commande sont indiqués entre crochets (<) et (>). Cela signifie qu'ils sont facultatifs.

Nous détaillons maintenant l'ensemble des actions élémentaires réalisées par le programme correspondant à la commande SIMULER.

Première étape: préparation de la simulation.

-chargement du modèle ou récupération du modèle courant.

-vérification de la liste des segments transposés d'entrée; ouverture de ces

segments; en fonction des dates fournies en paramètre, des dates des segments et du décalage maximum du modèle obtention de l'intervalle de temps sur lequel doit s'effectuer l'alimentation des variables.

-s'il y a des variables d'écart, traitement du segment d'entrée.

-initialisation des zones de travail en mémoire centrale correspondant aux paramètres et aux coefficients s'il y en a.

-s'il n'existe pas création du segment qui contiendra es valeurs des variables résultats de la simulation. -initialisation des zones de travail en mémoire centrale correspondant aux variables d'écart retardées.

-initialisation des zones de travail en mémoire centrale correspondants aux variables exogènes et endogènes retardées.

-sauvegarde de ce contexte de simulation.

Deuxième étape: calcul proprement dit et sauvegarde des résultats.

-boucle sur le nombre de périodes calculé précédemment, à chaque période on effectue les étapes suivantes:

alimentation de la zone de travail pour les variables endogènes et exogènes et les variables d'écart

calcul des valeurs des variables endogènes du modèle. Pour cela on effectue un certain d'itérations.

écriture des résultats dans le segment de résultats

décalage de la zone de travail

passage à la période suivante.

Troisième étape: fin de la tache de simulation.

- fermeture de tous les segments ouverts et libération de la place occupée.

CHAPITRE IV

MECANISMES D'EXTENSION ET DE MODIFICATIONS

1. Extension à partir du langage MODULECO

2. Extension par ajouts de modules écrits en PASCAL ou FORTRAN

3. La création de commandes

4. Mise à jour et modification d'une commande

Introduction

MODULECO a été conçu pour être un système évolutif, c'est à dire capable d'offrir de nouvelles fonctions au fur et à mesure de son développement. De plus, un utilisateur doit pouvoir définir des fonctions pour son propre usage, sans que celles-ci soient intégrées dans le système commun. Il doit pouvoir y accéder par une extension du langage de commande qui lui soit propre.

Pour cela, on a écrit des commandes permettant de créer, supprimer, mettre à jour des commandes aussi bien au niveau du système qu'au niveau de chaque utilisateur.

1. Extension à partir du langage MODULECO

1.1 La commande EDITCOM

La commande EDITCOM dont nous donnons la syntaxe plus loin permet d'entrer ou de modifier le texte d'un programme écrit en langage MODULECO. Appelée sans paramètre, elle place l'utilisateur en mode entrée de l'éditeur de texte MODULECO intégré au logiciel et lui permet ainsi d'entrer le texte d'un nouveau programme. Lorsqu'on spécifie le nom d'un programme c'est que l'on désire modifier ce texte aprés avoir fait des essais.

Syntaxe de la commande:

EDITCOM EDC
EDITION DE COMMANDES < NOMCOM CHAINE >
NOM DE LA COMMANDE

Dans le cas d'un nouveau programme, l'utilisateur en rentre le texte grâce à l'éditeur MODULECO. Ce texte est ensuite vérifié par appel à l'analyseur syntaxique du langage. Si aucune erreur n'est détectée, on passe la main au traducteur qui traduit ce programme en une forme interprétable. Si, au cours de ces deux étapes, des erreurs sont détectées, l'utilisateur repasse en mode édition de l'éditeur pour pouvoir corriger son texte. Pour sortir de la commande on tape fin si tout c'est bien passé ou sortir mais dans ce cas le travail effectué est perdu.

Une fois écrit un programme MODULECO peut être conservé sous la forme d'une commande qui vient ainsi s'ajouter à celles qui existent déja en s'insérant dans leur hiérarchie. Cette insertion est effectuée grâce à la commande CREER-COMMANDE (CCOM) dont nous parlerons plus loin. Cependant tout programme peut être simplement exécuté par appel direct au terminal. Seuls les programmes ayant donné lieu à la création de commandes sont sauvegardés d'une session à l'autre. Les autres sont détruits en fin de session.

1.2 Syntaxe du langage MODULECO

Eléments du langage:

Ce sont les identificateurs, les mots-clés et les caractéres spéciaux. Les identificateurs représentent des noms de variables, de constantes, de fonctions ou de procédures. Les mots-clés sont des mots réservés. Les caractères spéciaux (+ - * # < > ...) servent de délimiteur ou d'opérateur. Ils ont chacun une signification particulière sauf à l'intérieur d'une constante chaîne de caractère ou d'un commentaire.

Format:

Le format est libre. Aucune tabulation n'est imposée. Les caractères blancs ne sont pas significatifs. Le point virgule (;) sert de séparateur entre déclarations et entre instructions.

Structure d'un programme:

Un programme est composé:

-d'un en-tête,

-d'une suite de déclarations, éventuellement,

-d'une liste d'instructions.

L'en-tête spécifie le nom et les paramètres du programme, c'est à dire les données qui lui seront transmises ainsi que le type du résultat pour une fonction.

Les déclarations permettent de nommer et typer chacune des données utilisées dans le programme.

Les instructions définissent les actions qui seront exécutées par le programme au cours de son exécution.

Les déclarations de constantes élémentaires

Nous avons défini au chapitre précédent les différents types de données que manipule un programme MODULECO. On ne peut déclarer que des constantes de type entier, réel, logique, date et chaîne de caractères.

Exemple: const entier I=45, J=50

reel X=10e-2, PI=3.14116

date dd=#ii/70, logique b=vrai;

Ces déclarations suivent la syntaxe de la charte 9.

Les déclarations de variables de type élémentaire

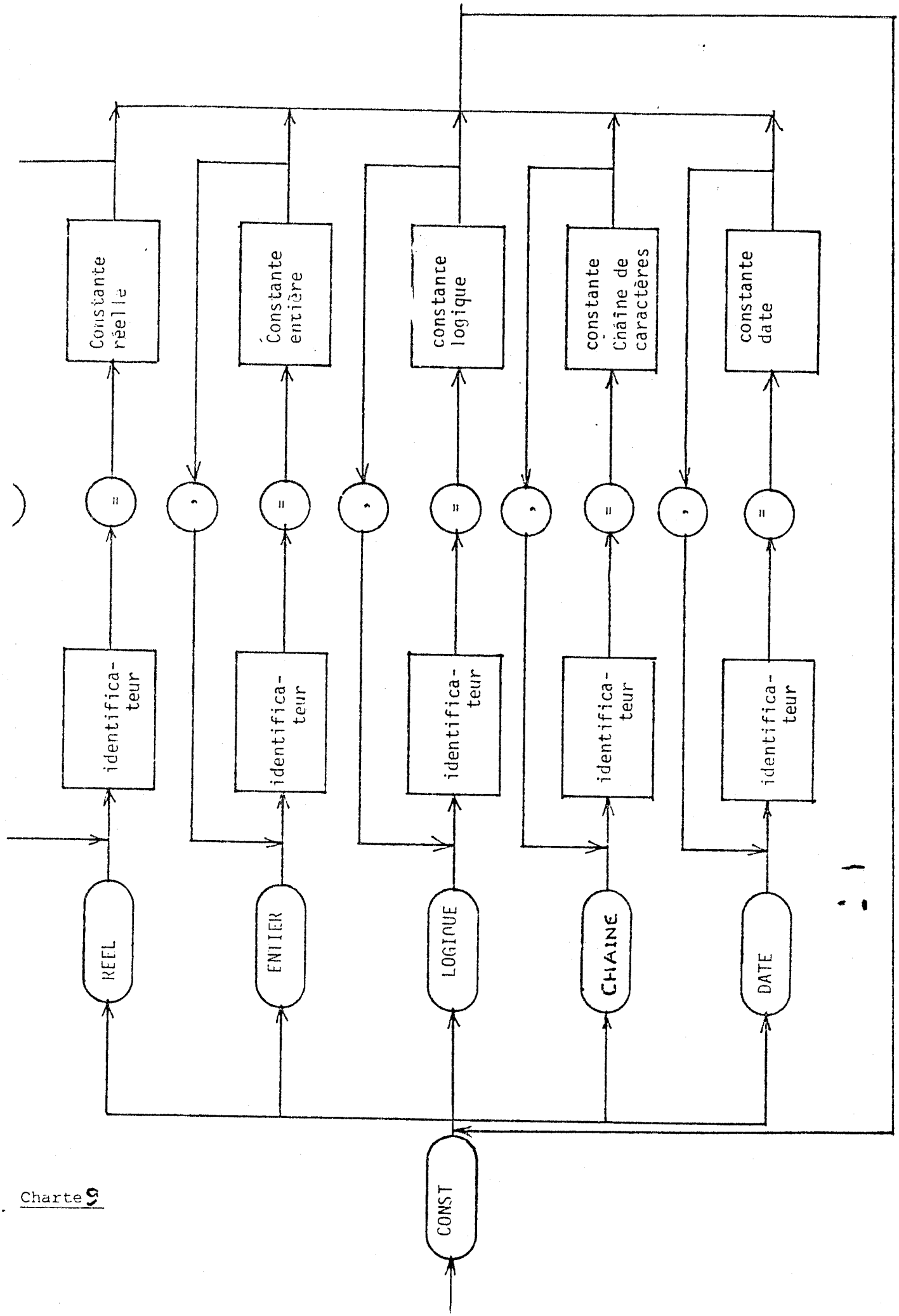
Elles sont déclarées selon la syntaxe de la charte 10. Toute déclaration de symbolique doit préciser un sous-type. Nous vous les avons détaillés plus haut.

Exemple: entier nboucle,taille;

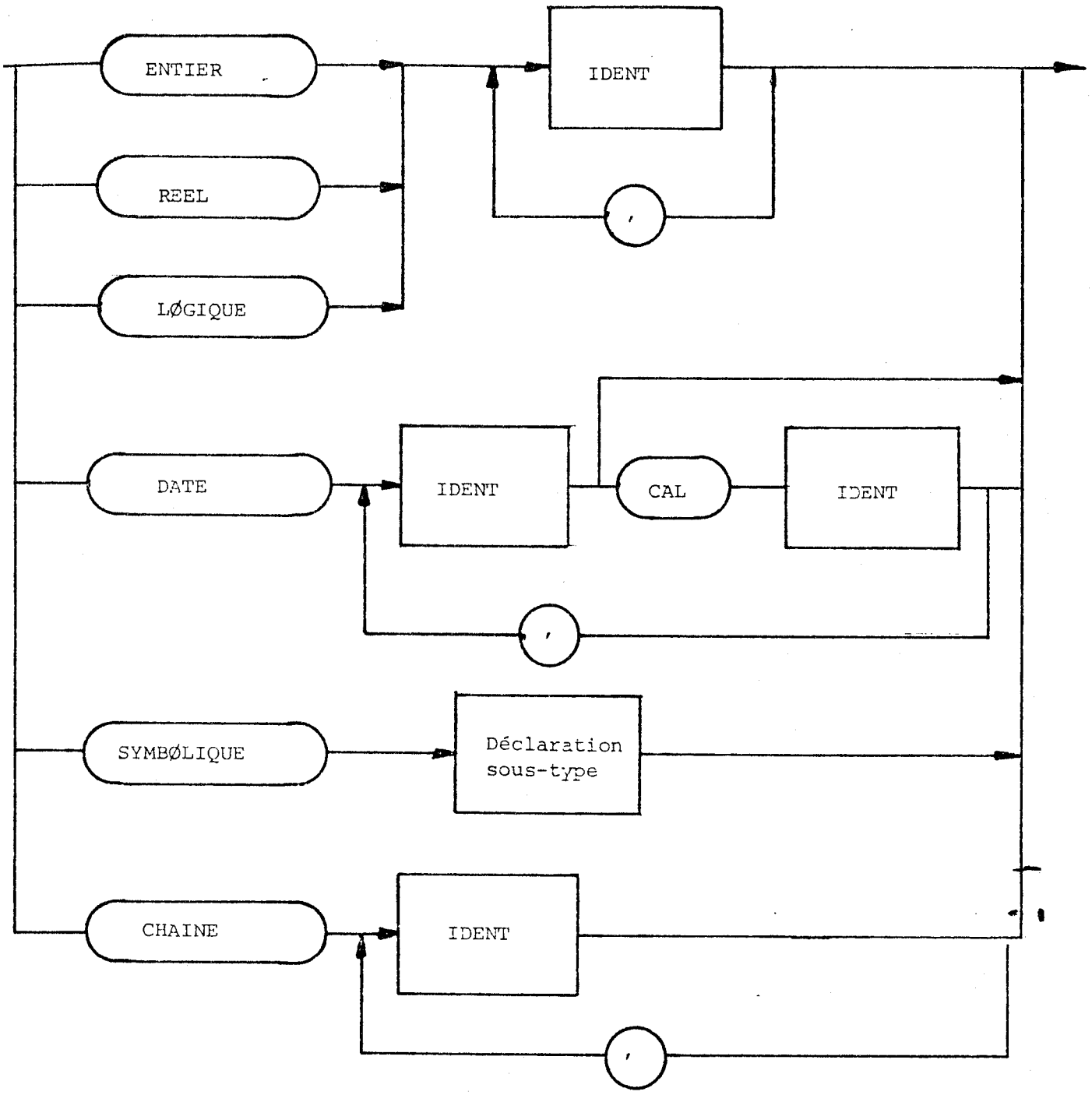
reel x,y,z;

logique trouve; chaîne titre;

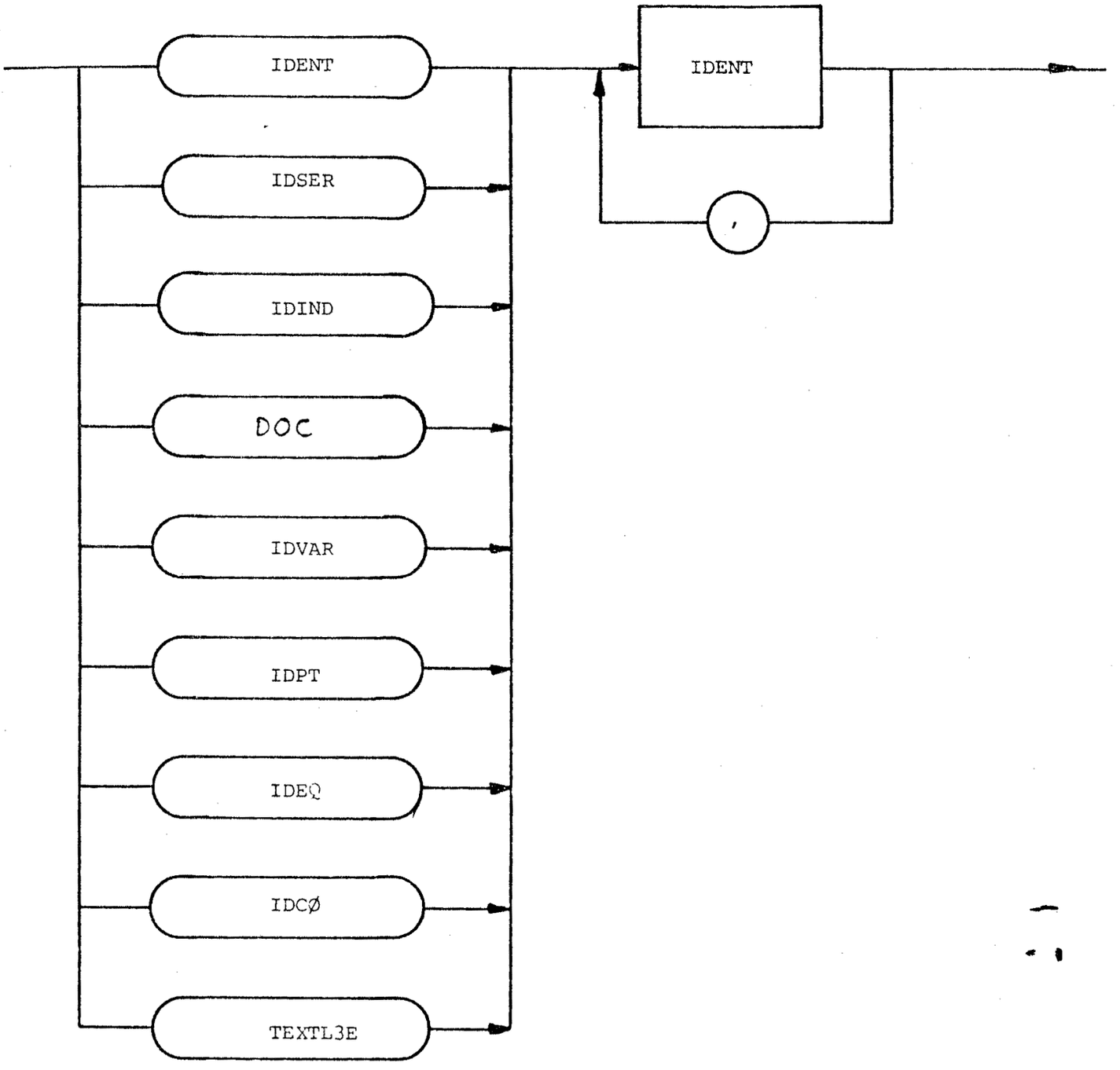
symbolique ident nmodele; date df cal trimestriel;


CONST
REEL
ENTIER
LOGIQUE
CHAINE
DATE
identifica-teur
identifica-teur
identifica-teur
identifica-teur
identifica-teur
=
=
=
=
=
,
,
,
,
Constante réelle
Constante entière
constante logique
constante Chaîne de caractères
constante date

Charte 9



Charte 10 - Déclaration variable de type élémentaire



Déclaration de sous-types

Constantes et variables listes homogènes

La syntaxe d'une constante liste homogène figure sur la charte 12.

Exemple: (#ii/70,#iii/70,#iv/71) constante liste de dates

(pnb, pib, cons) constante liste de symbolique ident

((1.2, 0.5), (-1E-2, 0.9), (1.1, 1.2, 1.3)) constante liste de réel à deux dimensions.

Les variables de type liste homogène sont déclarées suivant la syntaxe de la charte 13.

Exemple: liste de liste mat de reel;

liste agregat de symbolique ident;

liste datmaj de date;

Constantes et variables listes hétérogènes

Ces listes ne peuvent avoir qu'une dimension. La déclaration d'une constante liste hétérogène suit la syntaxe de la charte 14.

Exemple: ((PNB,1.2), (PIB,1.3), (TAUXV,1.5))

((CONS,4,VRAI), (LIQUID,10,FAUX))

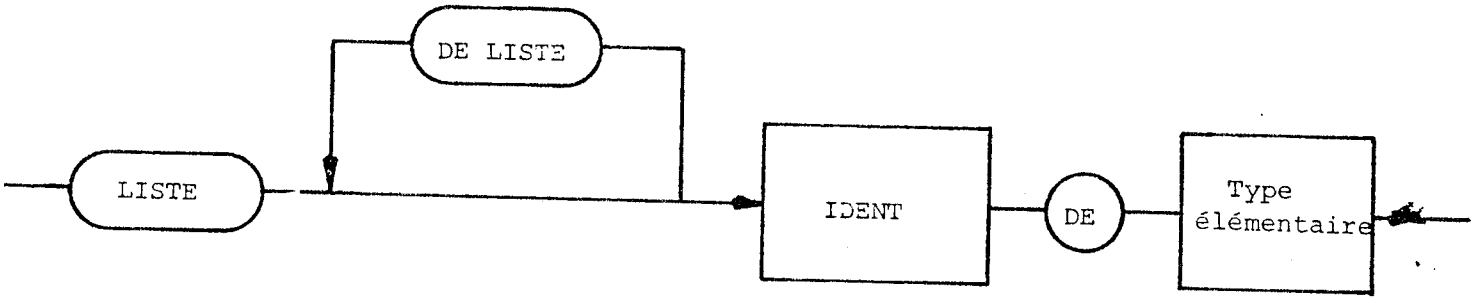
La liste vide se note: ().

La déclaration d'une variable liste hétérogène doit fournir le nom et le type de chaque champ et respecte la syntaxe de la charte 15 .

Exemple: liste vtest de (symbolique ident agregat; reel valtest);

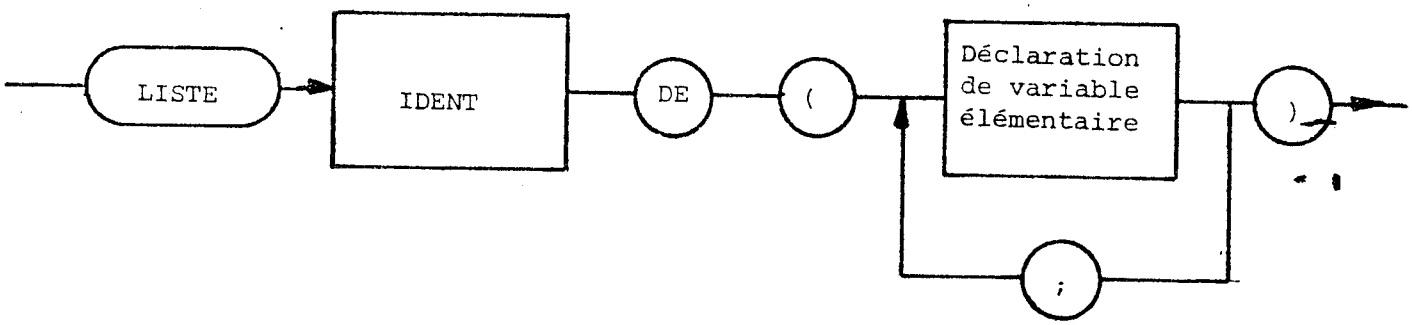
liste boucle de (symbolique ident agregat; entier nboucle; logique limite);

Déclaration de variable liste homogène



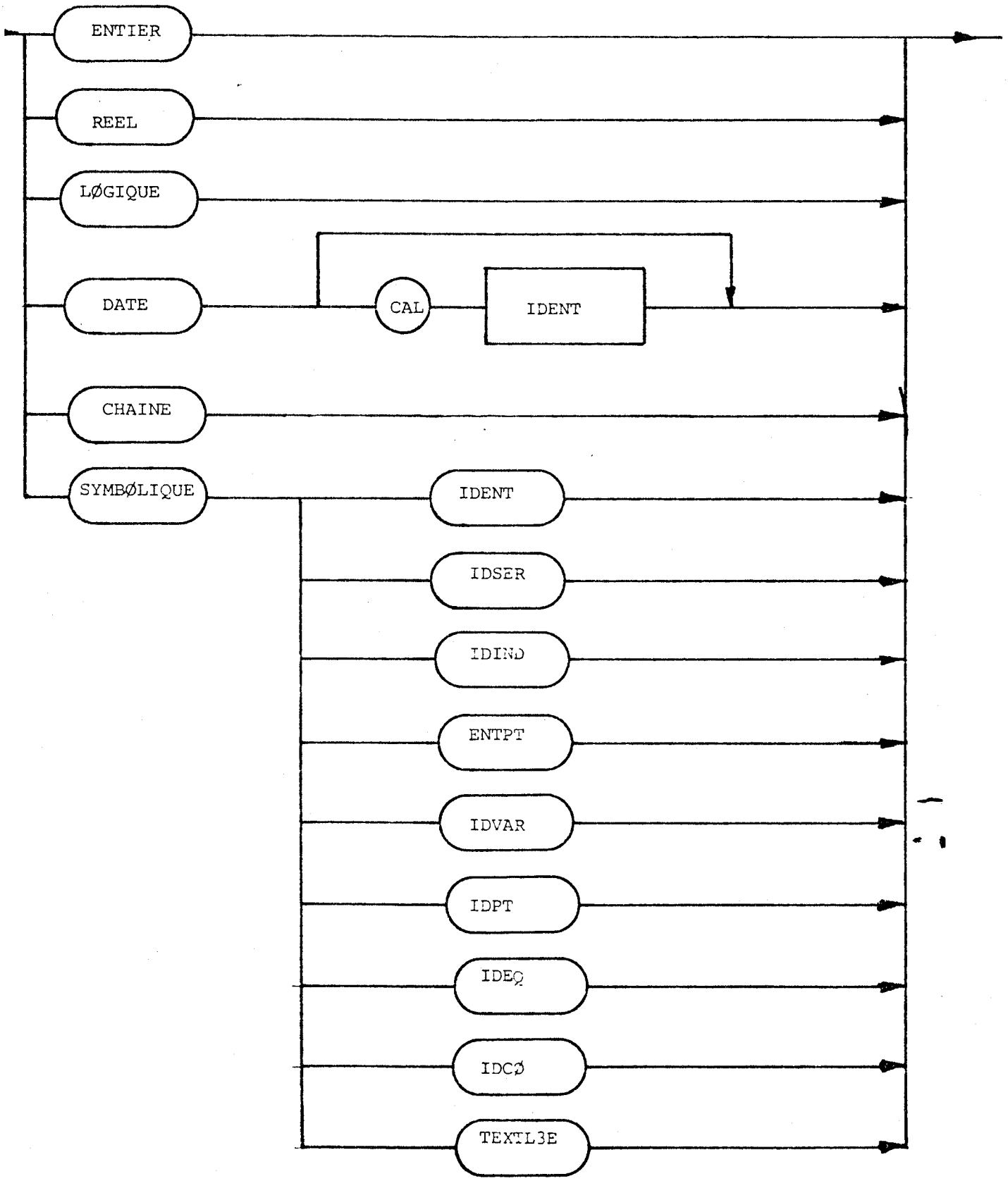
Charte 13

éclaration de variable liste hétérogène



Charte 15

ype élémentaire



Variables globales

Une variable dont la déclaration est précédée du mot clé GLOBAL est accessible à tous les programmes qui incluent la même déclaration. Elle peut y être utilisée et modifiée.

Les appels de procédures et fonctions

Ils ont la même syntaxe qu'ils se fassent directement au terminal ou soient insérés dans un programme. Le nom du programme appelé est suivi de la liste des arguments à lui transmettre.

La liste des arguments d'une fonction doit être donné entre parenthèses. Une fonction renvoie toujours un résultat. Il est imprimé au terminal dans le cadre d'une session interactive.

Exemple d' appels de procédure:

```
simuler lafay.oscillateur lientr=(donneest) coeff=resest %impvari;

grapht 1 (dataosc)
(cprr,impr,pnbn)
titg=('TROIS SERIES','ANNUELLES')
notg=('CES 3 SERIES APPARTIENNENT','A DONNEESOSC')
typetcp=((3,1,COURBE),(2,4,COURBE),(2,2,POINT))
typet=((ECHELY,5),(ECHELD,5))
epaitcp=((1,1.6,COURBE))
refd=(#1955,#1960,#1965)
margd=15.0
titecy=('MILLIARDS DE DOLLARS')
titecd=('TEMPS') titp=((3,#1963,'POINT 63'))
titry=((3,'COURBE DE PNBN'))
titrefd=((2,'NOUS SOMMES EN 1960'))
police=((TITG,2),(NOTG,8))
taille=((TITECD,3),(NOTG,4),(TITP,3),(TITREFD,3),(TITG,8),
```

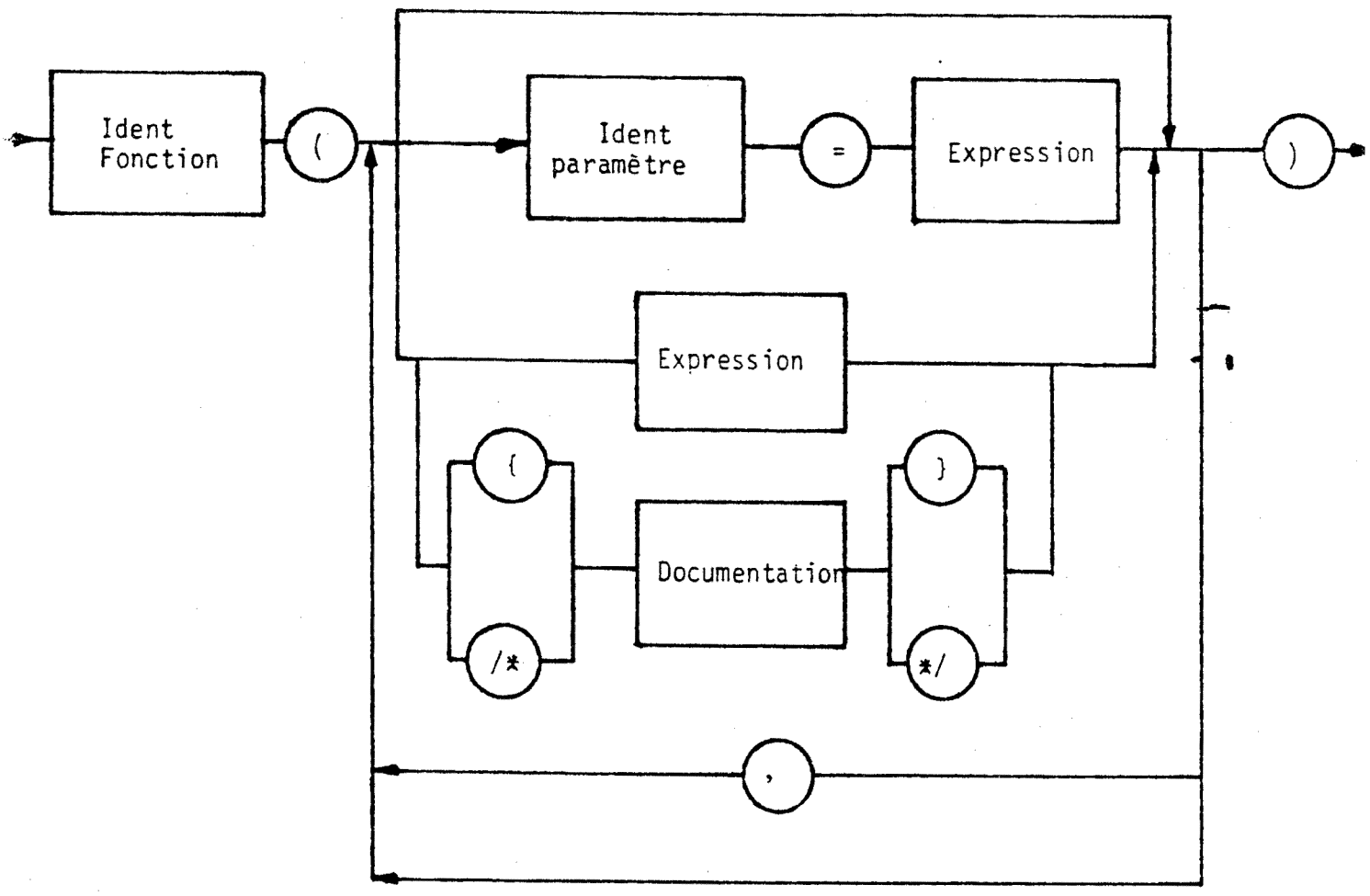
(TITECY,3))
position=((TITECY,HAUT,GAUCHE),(TITECD,HAUT,DROITE),
(NOTG,BAS,GAUCHE),(TITG,HAUT,GAUCHE))
rot=(TITECY) epait=((TITG,1.2))
refy=(200,400,600) titrefy=((2,'SEUIL DES 400 MILLIARDS'));

appels de fonction:

liexo;

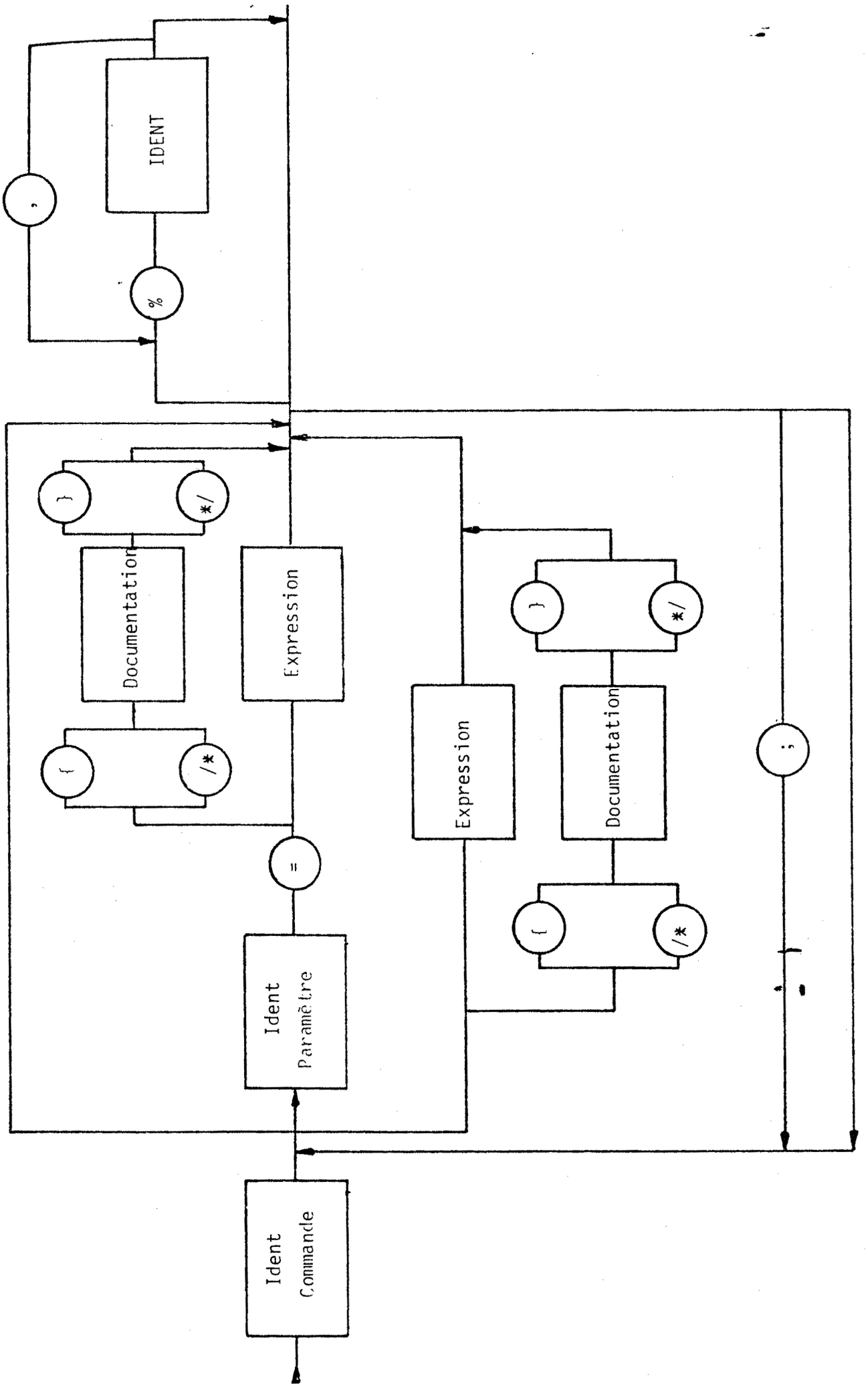
liendo (bloc1);

Appel Fonction



Charte 17

Appel Commande Charte 17



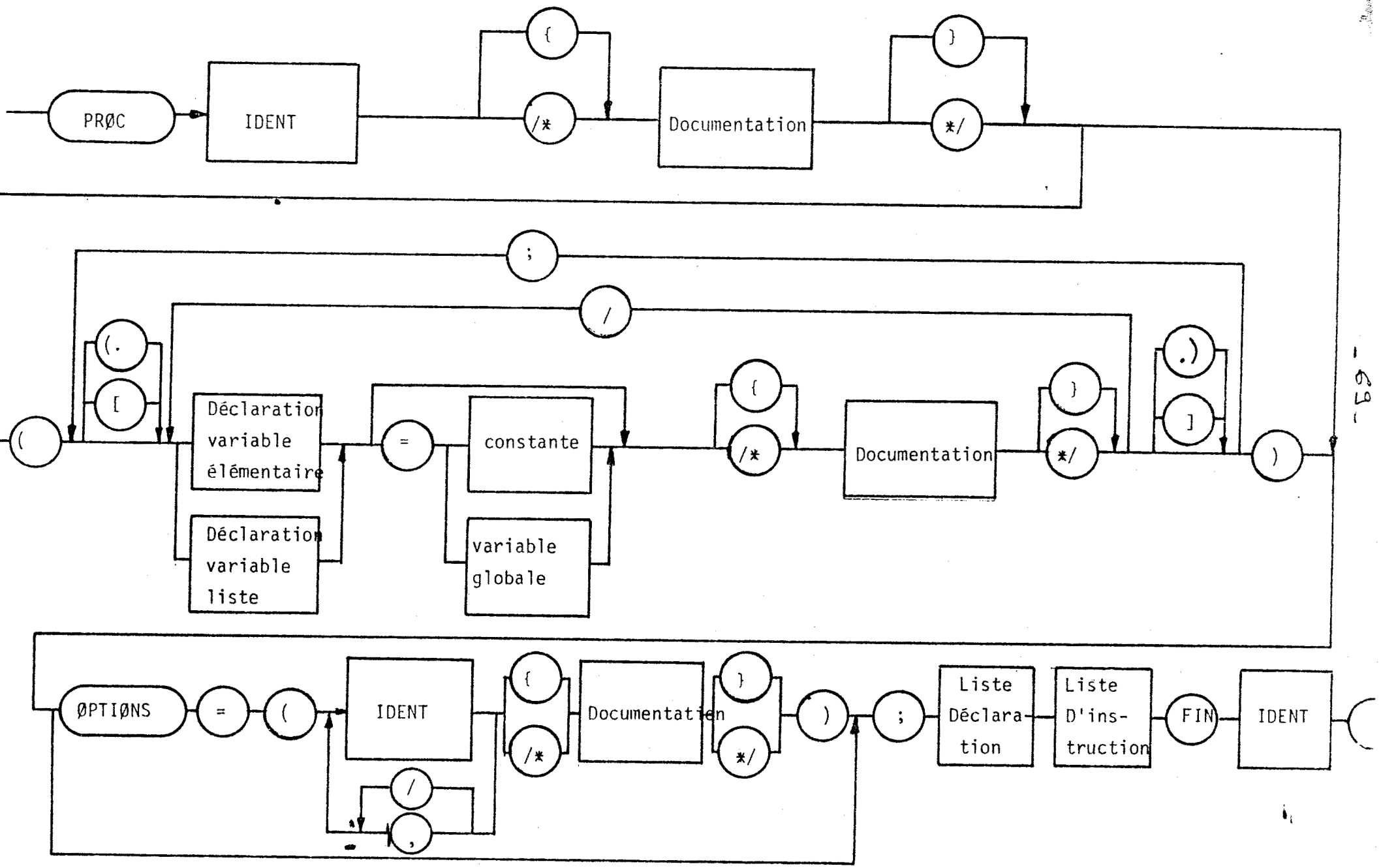
Les en-têtes de programme

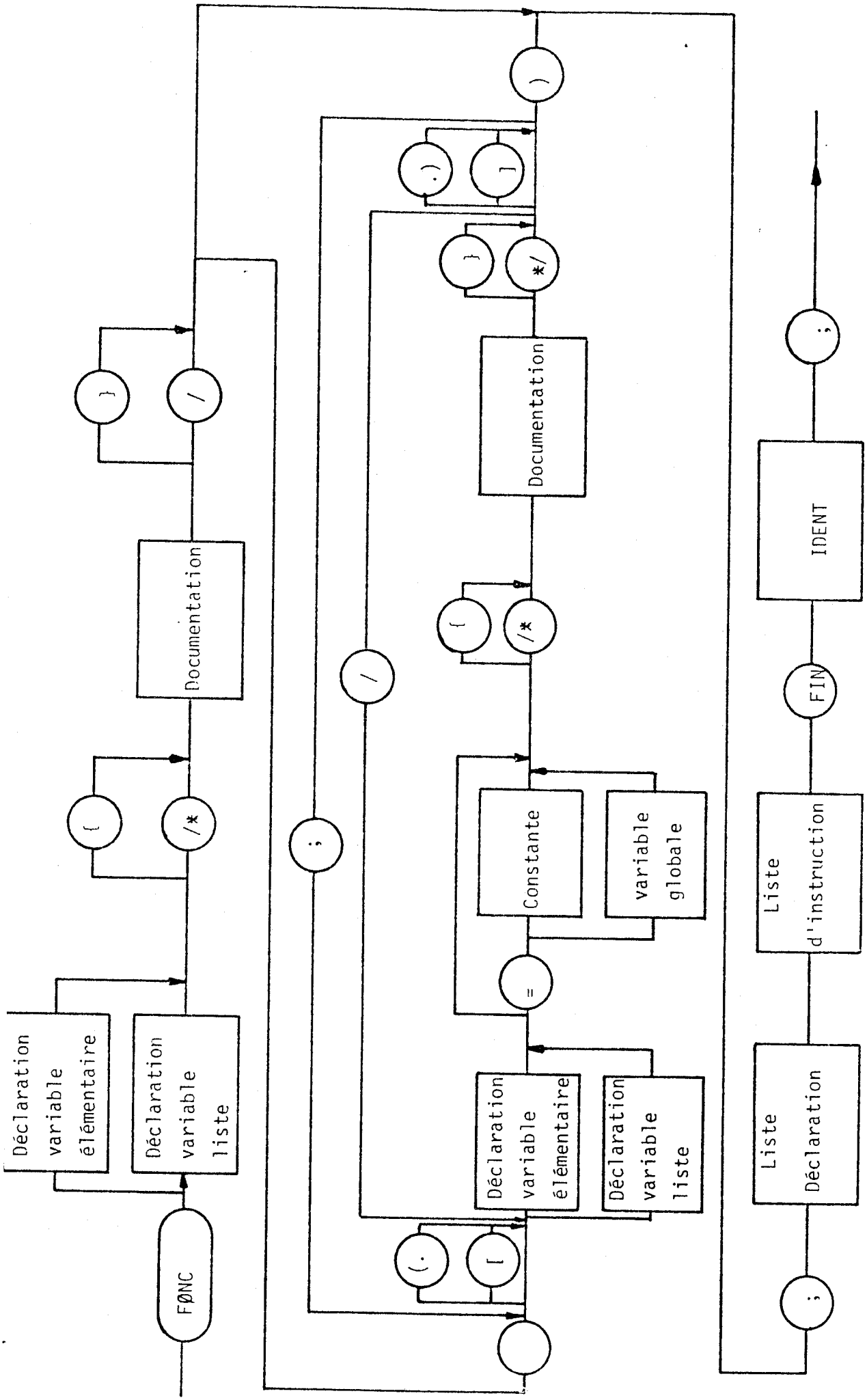
Ils fournissent le nom et la documentation du programme, le nom et le type de chacun des paramètres, leur valeur par défaut et leur documentation. Une valeur par défaut est donnée à la suite du nom du paramètre auquel elle se rapporte sous forme d'une constante précédée du signe égal. Pour une liste la seule valeur par défaut possible est la liste vide.

Un paramètre placé entre crochets (<) et (>) est facultatif. Si deux paramètres au lieu d'être separe par un point virgule (;) le sont par un slash (/) cela veut dire qu'ils sont mutuellement exclusifs.

Exemple:

```
proc transposer /*transposition
option nonexo : pas de transposition des variables exogenes */
(<symbolique idpt mod /*nom du modele*/>;
<liste libloc de symbolique ident /*liste des blocs */>;
liste liseg de symbolique idpt /*liste des segments de series*/;
symbolique ident segst /*segment transpose cree*/;
<date dd /*date de debut*/>;<date df /*date de fin*/>;
<liste liser de symbolique ident /*liste des series a transposer*//
liste lisauf de symbolique ident /*liste des series a ne pas transposer*/>;
<liste linom de (symbolique ident nvar ; symbolique idser nser )>
/*correspondance nom de variable-nom de serie*/);


fonc liste liboucl de symbolique ident
/* liste des variables de bouclage du modele courant */
(<liste libloc de symbolique ident /* liste des blocs */>);
```


PRØC
IDENT
{
/*
Documentation
}
*/
;
/
(.
[
(
Déclaration variable élémentaire
Déclaration variable liste
=
constante
variable globale
{
/*
Documentation
}
*/
.)
]
)
ØPTIØNS
=
(
IDENT
/
,
{
/*
Documentation
}
*/
)
;
Liste Déclaration
Liste D'instruction
FIN
IDENT


FØNC
Déclaration variable élémentaire
Déclaration variable liste
Documentation
Constante
variable globale
Déclaration variable élémentaire
Déclaration variable liste
Documentation
Liste Déclaration
Liste d'instruction
IDENT
FIN

Charte 18

Les instructions

L'affectation

La variable en partie gauche et l'expression en partie droite doivent être de même type. Pour des symboliques, il doit y avoir de plus identité des sous-types.

Les expressions font intervenir des variables de type élémentaire. Les opérateurs arihmétiques portent sur des valeurs entières et réelles. Ce sont les opérateurs classiques + - * / ** et les opérateurs DIV (division entière) et MOD (reste de la division entière). Les opérateurs logiques portent sur des valeurs logiques et sont & (et logique), | (ou logique), (négation). Les opérateurs relationnels portent sur tous les types et renvoient une valeur logique. Ce sont <, >, <=, >=, =, <>. L'opérateur relationnel DANS fait intervenir les types structurés et permet de vérifier si une valeur donnée apparaît ou non dans une liste. L'opérateur de concaténation || porte à la fois sur les symboliques et les chaînes.

Les expressions s'évaluent de gauche à droite. Les priorités des opérateurs données en ordre croissant sont les suivantes:

< > = <> <= >= DANS
+ - | ||
* / &
-i + - (unaires)
**

Exemple: avec les déclarations suivantes

```
entier i; reel x,y,z,u,v; logique a; symbolique ident nommod;

i := i + 1;

a := ((i mod 2) = 0);

b := ((x + y + z) <= (u/v));
```

nommod := metric; pa +1

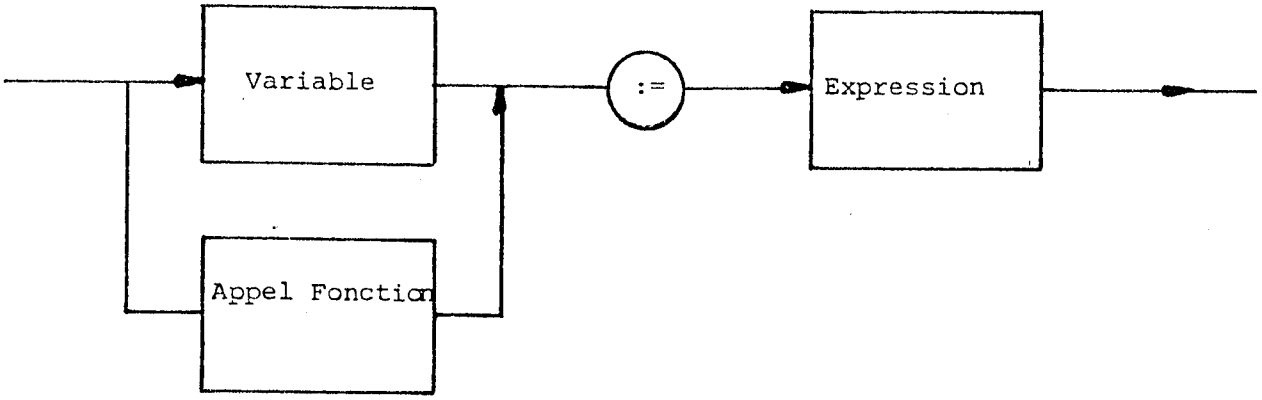
Les instructions de contrôle

Elles permettent de modifier la séquence stricte dans laquelle s'exécute un programme à partir de la première instruction. On distingue les instructions d'exécution conditionnelle (SI, CHOIX) et les instructions itératives (TANTQUE, REPETER, POUR).

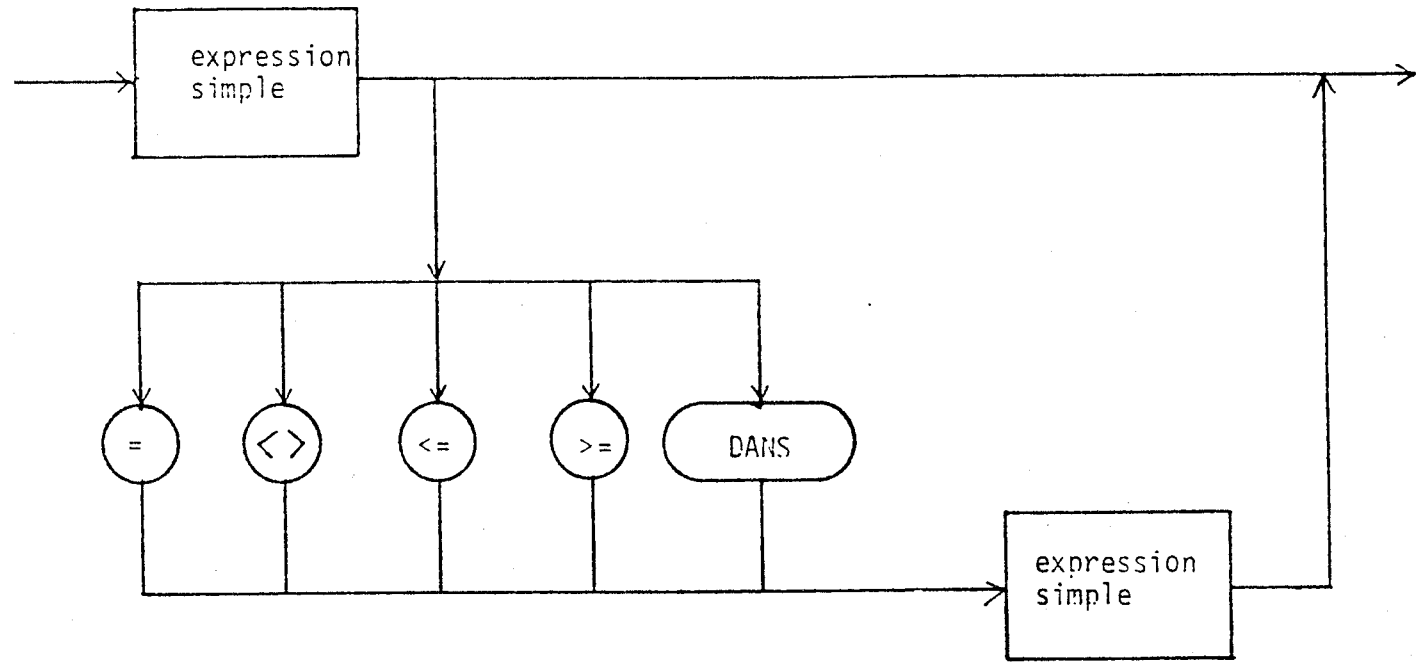
Exemple:

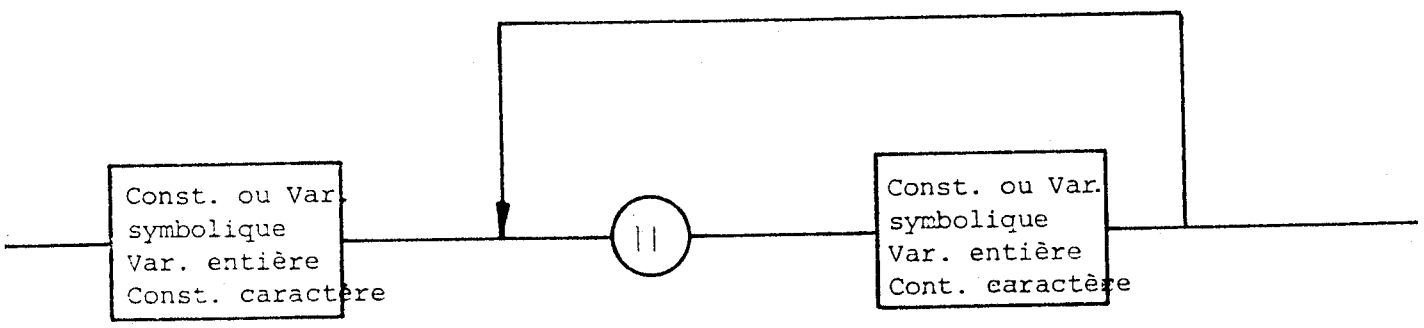
```
si x < y alors max := y
      sinon max := x fin;

i:= 1;
tantque nomvar <> lvar(i) faire i := i + 1;

repeter gausseidel
jusqua (iter >= 50) ou (noconver=());
```

Affectation



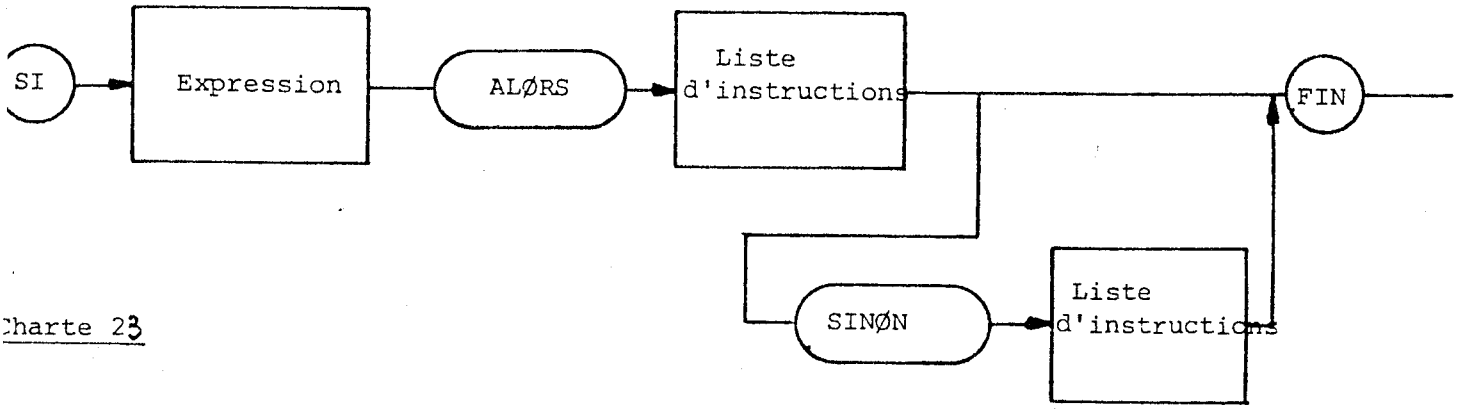
expression





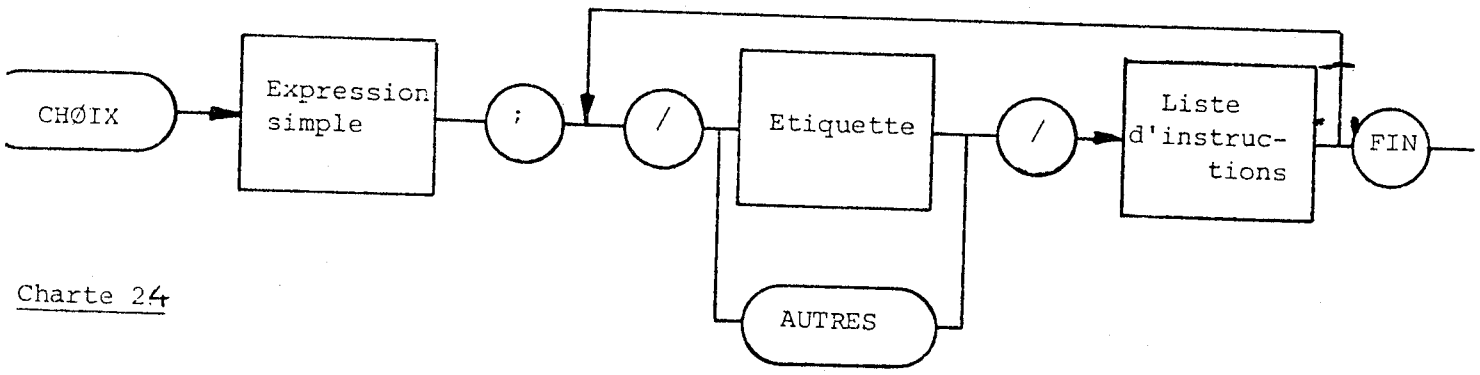
Expression symbolique

Instruction SI



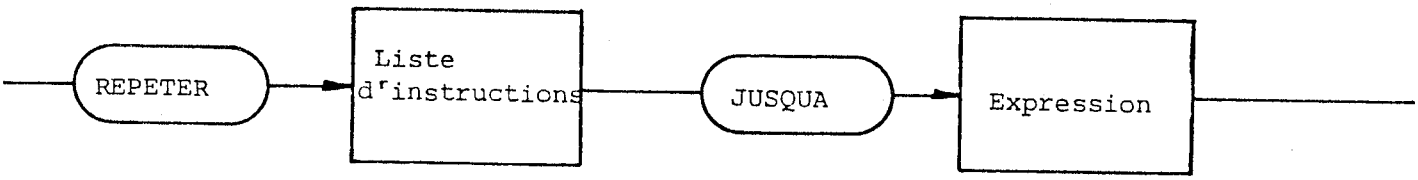
Charte 23

Instruction CHØIX

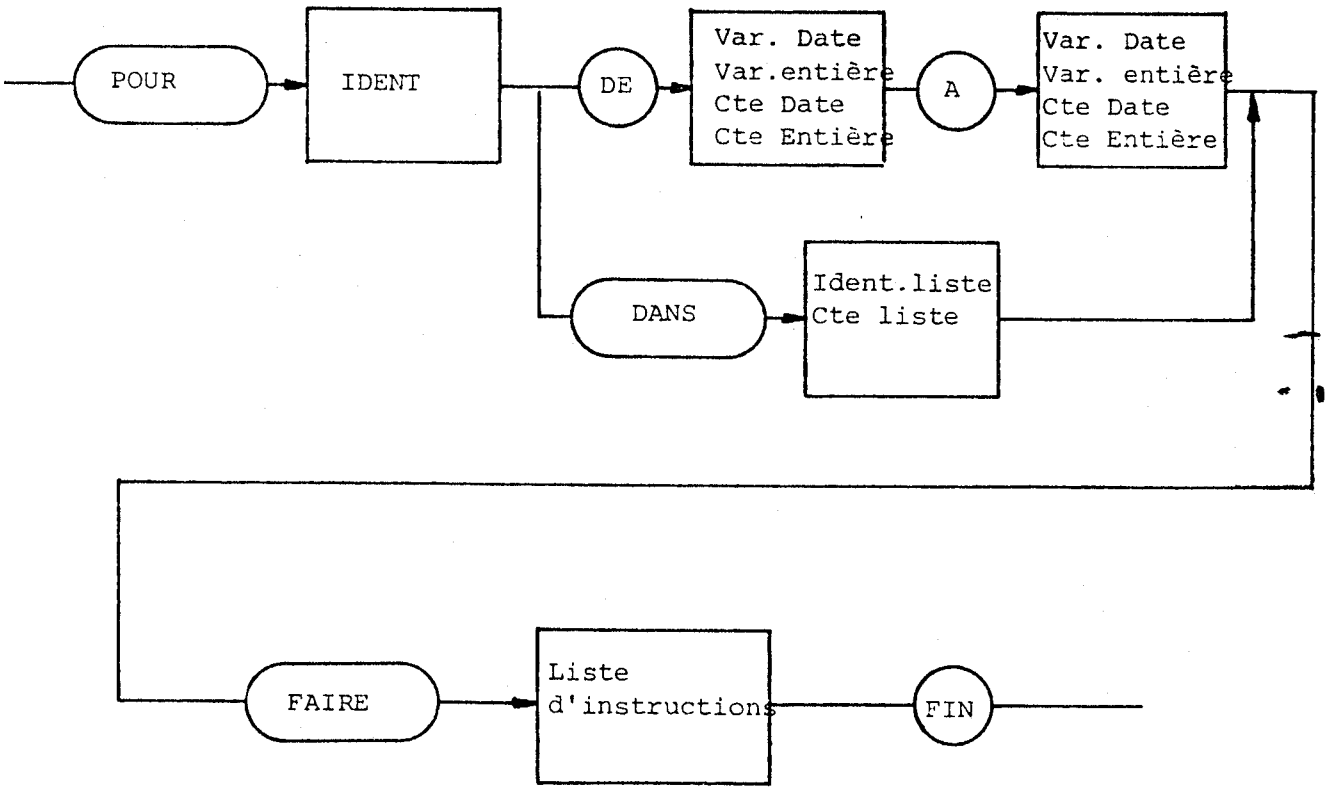


Charte 24

Charte 26 - Instruction TANTQUE



Charte 27 - Instruction REPETER



Charte 28 - Instruction POUR

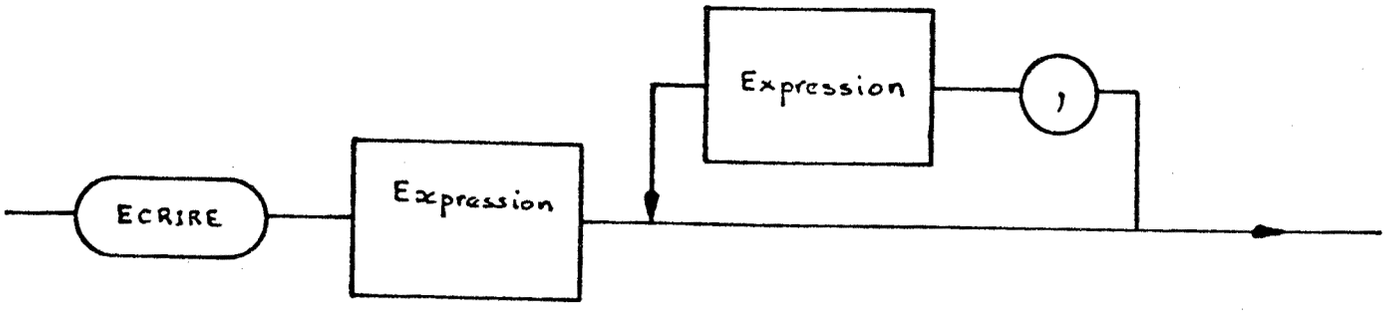
Les entrées/sorties au terminal

ECRIRE permet d'imprimer au terminal les valeurs de constantes variables simples ou structurées dont les noms sont spécifiées. Le format d'impression suit la syntaxe du langage.

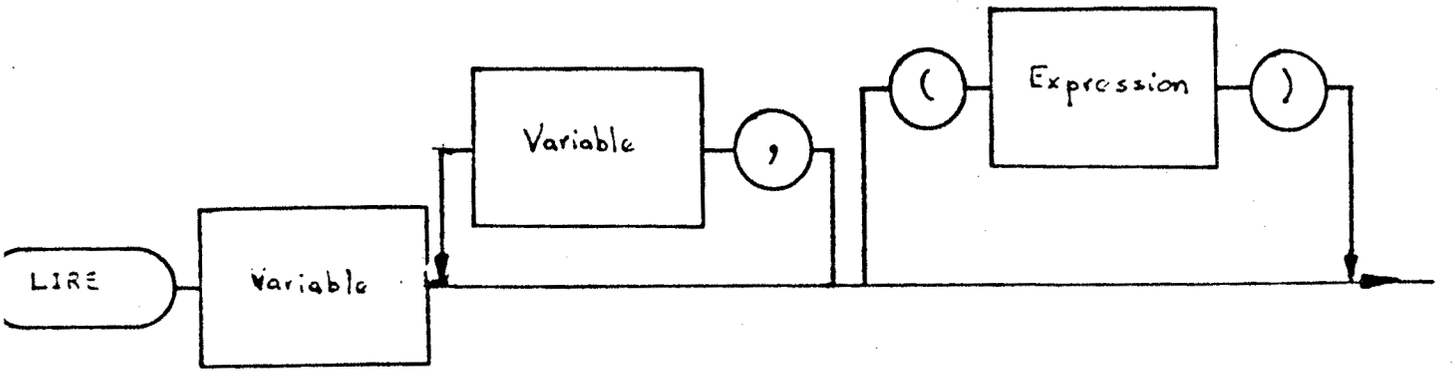
ECRIRELG permet de passer à la ligne.

LIRE permet de lire au terminal les valeurs de variable de tout type.

Exemple: Ecrirelg 'nom de la variable ?';
lire nomv;



Charte 30



Charte 31

Voici des exemples d'utilisation du langage de commandes. Le premier exemple montre comment on peut enrichir le logiciel en créant des programmes qui se composent d'appels à des commandes déja existantes. Il s'agit d'offrir aux utilisateurs une nouvelle méthode d'estimation, les doubles moindres carrés (DMC).

un modèle linéaire à équations simultanées s'écrit sous la forme standard:

$$(1)\quad Y_i = a_{12}Y_2 + a_{13}Y_3 + \ldots + a_{1n}Y_n + b_{11}X_1 + \ldots + b_{1p}X_p + \varepsilon_1$$

...

$$(n)\quad Y_n = a_{n1}Y_1 + a_{n2}Y_2 + \ldots + a_{nn-1}Y_{n-1} + b_{n1}X_1 + \ldots + b_{np}X_p + \varepsilon_n$$

ou en abrégé l'équation (i) s'écrit encore:

$$(i)\ Y_i = \sum_{\substack{j=1 \\ j!=i}}^{n} a_{ij}Y_j + \sum_{k=1}^{p} b_{ik}X_k + \varepsilon_i$$

où les $(Y_i)_{i=1,n}$ sont les variables endogènes;

les $(X_j)_{j=1,p}$ sont les variables exogènes;

les $(epsilon_i)_{i=1,n}$ sont les aléas.

Une équation sera dite

-identifiable si le nombre d'endogènes et d'exogènes absentes de cette équation est égal au nombre total d'endogènes du modèle diminué de un,

-suridentifiable si le nombre de variables absentes excède le nombre d'endogènes diminué de un.

Pour ces équations la méthode naïve consistant à régresser chaque équation (i) par les moindres carrés ordinaires (MCO) est discutable. on recommande la méthode des doubles moindres carrés définie ainsi:

pour chacune de ces équations

1. on remplace chaque endogène Y_j apparaissant comme explicative par sa valeur ajustée $\hat{Y}_j$ obtenue par régression sur l'ensemble des exogènes du modèle

2. on régresse l'équation obtenue par substitution des $\hat{Y}_j$ précédents aux Y_j voulus.

Texte de la commande:

```
proc dmc
/* estimation par la méthode des doubles moindres carrés
 de l'équation spécifié par la commande mère estimer */
(liste livar de symbolique idvar
/* liste des endogènes apparaissant comme explicatives */;
liste segser de symbolique idpt
/* liste des segments de séries */);

entier i;
liste linouvseg de symbolique ident;

/* régression des endogènes en fonction des exogènes */
i := 1;
tantque livar <> () faire

/* régression de chaque endogène de la liste livar
 en fonction de toutes les exogènes du modèle */
  reglin livar(i),liexo,segser;

/* la valeur estimée de cette endogène est
 stockée dans le segment de série dmc */
  stockval dmc,livar(i);
  i := i + 1;
fin tantque;
finest; /* fin du contexte d'estimation courant */
linouvseg(1) := ' estim.dmc';
segser := concat (segser,linouvseg);
```

```
/* on définit un nouveau contexte d'estimation */
estimer liseg = segser;

/* l'équation est réestimée par les mco; les endogènes apparaissant
comme explicatives sont remplacées par leur valeur ajustée */
mco nomest := mco;

/* on trouvera dans le segment de coefficients
 'mco' le résultat de cette estimation */
fin dmc;
```

Voici un autre exemple de programme écrit en langage MODULECO:

```
proc gestdon /* gestion de paramètres */;

symbolique idpt s1;
symbolique ident idser;
date d1,d2,dinter;

ecrirelg 'nom du segment de travail ?';
lirelg s1;
ecrirelg 'calendrier ?';
lirelg idper;
ecrirelg 'date de debut ?';
lirelg d1;
ecrirelg 'date de fin ?';
lirelg d2;
si d2 < d1 alors
  dinter := d1;
  d1 := d2;
  d2 := dinter;
fin si;
initseg s1,d1,d2,idper;
fin gestdon;
```

2.Extension par ajout d'un module écrit en PASCAL et en FORTRAN

En Décembre 1984, le logiciel a été enrichi d'un ensemble de commandes graphiques:

GRAPHT pour tracer des séries temporelles,

GRAPHC pour tracer des séries croisées,

PGRAPH pour mettre en page des objets graphiques.

GRAPHT permet de tracer jusqu'à six séries temporelles élémentaires sur un même référentiel. Le graphique se trace plein écran si la visualisation est demandée; s'il est stocké, on en obtiendra la visualisation par PGRAPH. GRAPHC permet de définir des graphiques de séries croisées Y fonction de X. PGRAPH permet de faire de la mise en page d'objets graphiques. A chaque objet graphique correspond un segment MODULECO qui contient les tables générées par l'analyse syntaxique et sémantique de tous les paramètres de la commande (GRAPHC ou GRAPHT) exécutée au cours d'une session antérieure.

A chacune de ces trois commandes correspond un module PASCAL. L'ensemble des sous-programmes qui le compose effectue une analyse sémantique des valeurs des paramètres. A partir de ces valeurs, des tables sont constituées, dans le cas de GRAPHC et GRAPHT. Si l'utilisateur le désire elles peuvent être stockées dans un segment MODULECO pour être exploitées dans une autre session par PGRAPH.

Un ensemble de routines écrites en FORTRAN 3D réalise les fonctions suivantes:

définition de le fenêtre sur l'écran, des marges et des axes sur la fenêtre

tracé des axes et des références verticales, horizontales et obliques

recherche des valeurs de référence optima et visualisation de ces valeurs sur

l'échelle des y et sur l'échelle des x (pour GRAPHC)

visualisation des valeurs des références et des dates sur l'échelle des temps (pour GRAPHT)

tracé des courbes

tracé des hachurages

édition des textes tracés avec une police de caractères

tracé des légendes

Ces commandes graphiques sont trés complètes et offrent de nombreuses possibilités. Voici une description de la commande GRAPHT et quelques exemples d'utilisation.

```
proc grapht
/*visualisation de séries temporelles
option nvisu   : le graphique ne sera pas visualisé
     nleg      : la légende ne sera pas visualisée
     autod     : références verticales de chaque point de l'échelle
                 des temps
     autoy     : références horizontales de chaque point de l'échelle
                 des y
     autoprd   : références verticales de chaque point de la courbe
     autopry   : références horizontales de chaque point de la courbe
     trstrd    : l'axe des temps n'est pas tracé
     trsty     : l'axe des y n'est pas tracé
     slog      : échelle semi-logarithmique*/
(<entier affunit
/*entier précisant le type de terminal graphique*/>;
liste liseg de symbolique idpt
/*liste des noms de segments*/;
liste liser de symbolique idser
/*liste des noms de séries*/;
<liste titg de chaîne
```

```
/* titre chaque chaîne représente une ligne de titre */>;
<liste notg de chaîne
/* note : avec une chaîne par ligne de la note */>;
<liste de liste lidate de date
/*il faut une liste par série  mais si les listes sont identiques
 pour toutes les séries une seule liste suffit
 la kième liste donne pour la kième série
   -la date de début de visualisation
   -la date de fin de visualisation
 ces dates sont exprimées dans le calendrier de la série
correspondante
 ( les années sont écrites avec leurs millésimes )*/>;
<liste pasec de entier
/* liste d'entiers qui donnent pour chaque sous ensemble
  du calendrier fusionné , le pas de progression pour
  la visualisation des dates sur l'échelle des temps */>;
<reel ydeb
/* borne inférieure sur l'axe des y */>;
<reel yfin
/* borne supérieure sur l'axe des y */>;
<liste echey de reel
/*liste des valeurs à visualiser sur l'axe y*/>;
<reel pasey
/*pas de progression pour écrire les valeurs sur l'échelle y*/>;
<liste typetcp de (entier numser1;entier nb;symbolique ident nat1)
/* liste hétérogène qui donne le type de tracé pour les courbes et
   points des courbes
chaque liste est constituée de :
     -un entier donnant le numéro d'ordre de la série
     -entier donnant le type de tracé
     -symbolique ident spécifiant si  courbe  ou  point */>;
<liste typet de (symbolique ident nat2;entier tc)
/* type de trace pour l'armature du graphique,
   liste hétérogène de :
    -symbolique ident définissant l'objet à tracer
    (echely,echeld,refy,refd,refo,autorefy,autorefd,autopry,autoprd)
    -entier donnant le type de tracé */>;
<liste epaitcp de (entier numser2;reel epais1;symbolique ident nat3)
/* épaisseur des courbes et des points,liste hétérogène de :
   -entier = numéro de série
   -réel   = épaisseur ( en mm )
```

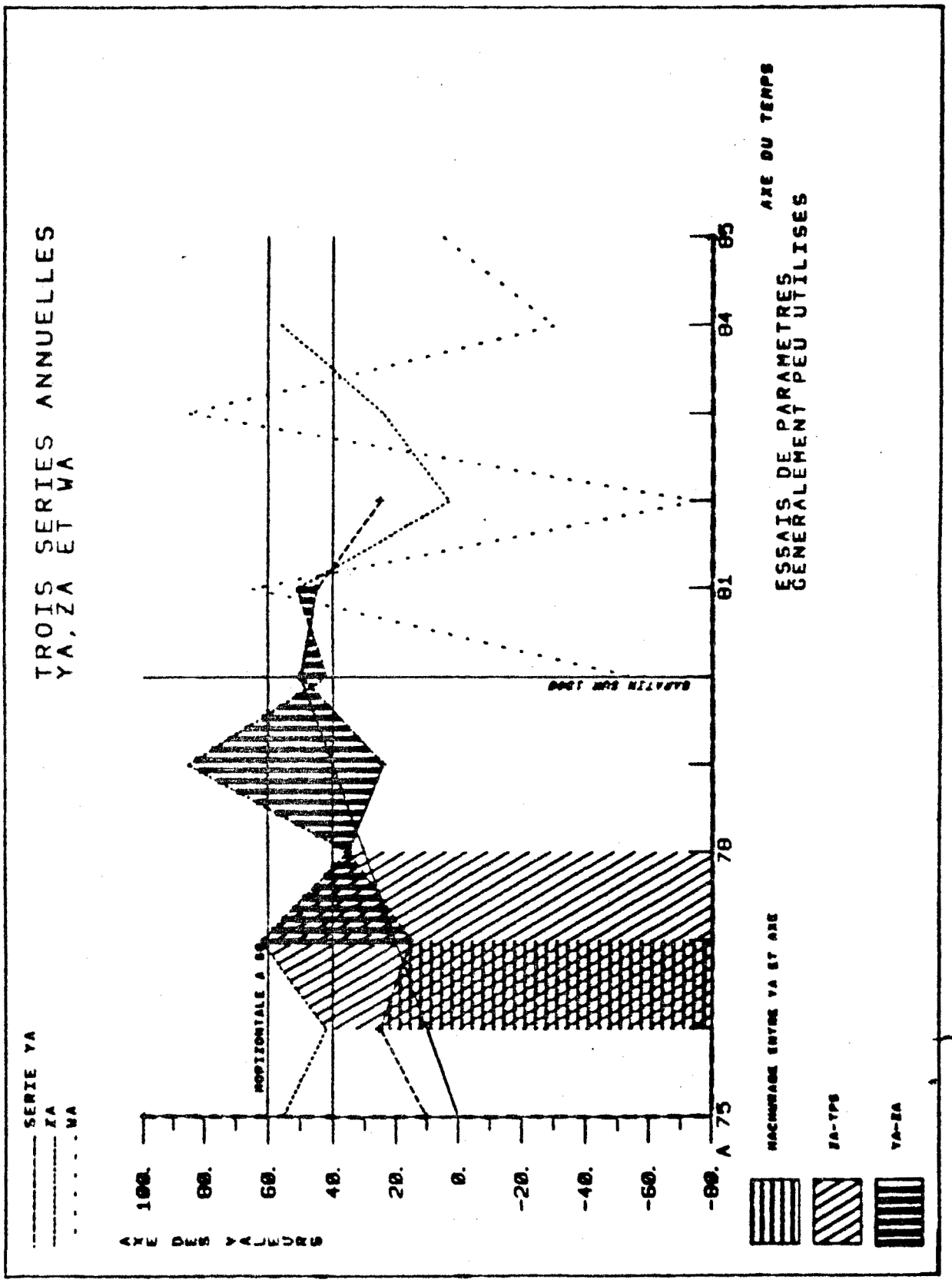
```
   -ident = courbe ou point */>;
 <liste epait de (symbolique ident nat4;reel epais2)
 /* épaisseur de l'armature du graphique,liste hétérogène de :
   -ident définissant l'objet à épaissir
    (echely,echeld,refy,refd,refo,autoref(y,d),autopr(y,d) )
   -réel donnant l'épaisseur en mm (par défaut 0.2mm) */>;
 <liste coultcp de (entier numser3;symbolique ident couleur;
 symbolique ident nat5)
 /* couleur des courbes et des points,liste hétérogène de :
   -entier = numéro d' ordre de la série dans liser
   -ident = couleur (rouge bleu jaune violet vert . . .)
   -ident = courbe ou point */>;
 <liste de liste coult de symbolique ident
 /* liste de liste homogène donnant la couleur de l'armature
  chaque liste est constituée de 2 symboliques idents
   -ident définissant la partie à colorier
    (echel(y,d),ref(y,d,o),autoref(y,d),autopr(y,d),titg,notg,titecy
    ,titecd,tity,titp,tith,titrefo,titrefd,titrefy)
   -ident couleur */>;
 <liste intcp de (entier numser4;entier int1;symbolique ident nat6)
 /* liste hétérogène donnant l'intensité des courbes et des points
  chaque liste est constituée de :
   -entier = numéro d'ordre de la série
   -entier = valeur de l'intensité ( de 0 a 5 )
   -ident = courbe ou point */>;
 <liste intt de (symbolique ident nat7;entier intt2)
 /* intensité de l'armature du graphique,liste hétérogène de :
  -ident spécifiant la partie du graphique
   (echel(y,d),ref(y,d,o),autoref(y,d),autopr(y,d),titg,notg,titecy,
    titecd,tity,titp,tith,titrefo,titrefd,titrefy)
  -entier donnant l'intensité ( de 0 a 5 ) */>;
 <liste refd de date
 /* liste de dates pour tracer des droites de référence verticales
  ces dates ne peuvent être que mensuelles ou annuelles
  pour cela il faut que parmi les séries il en existe une
  mensuelle ou une annuelle
  le 'ou' n'est pas exclusif
  des dates annuelles et mensuelles peuvent coexister dans la liste
 */>;
 <liste refy de reel
 /*liste de réels pour tracer des références horizontales*/>;
```

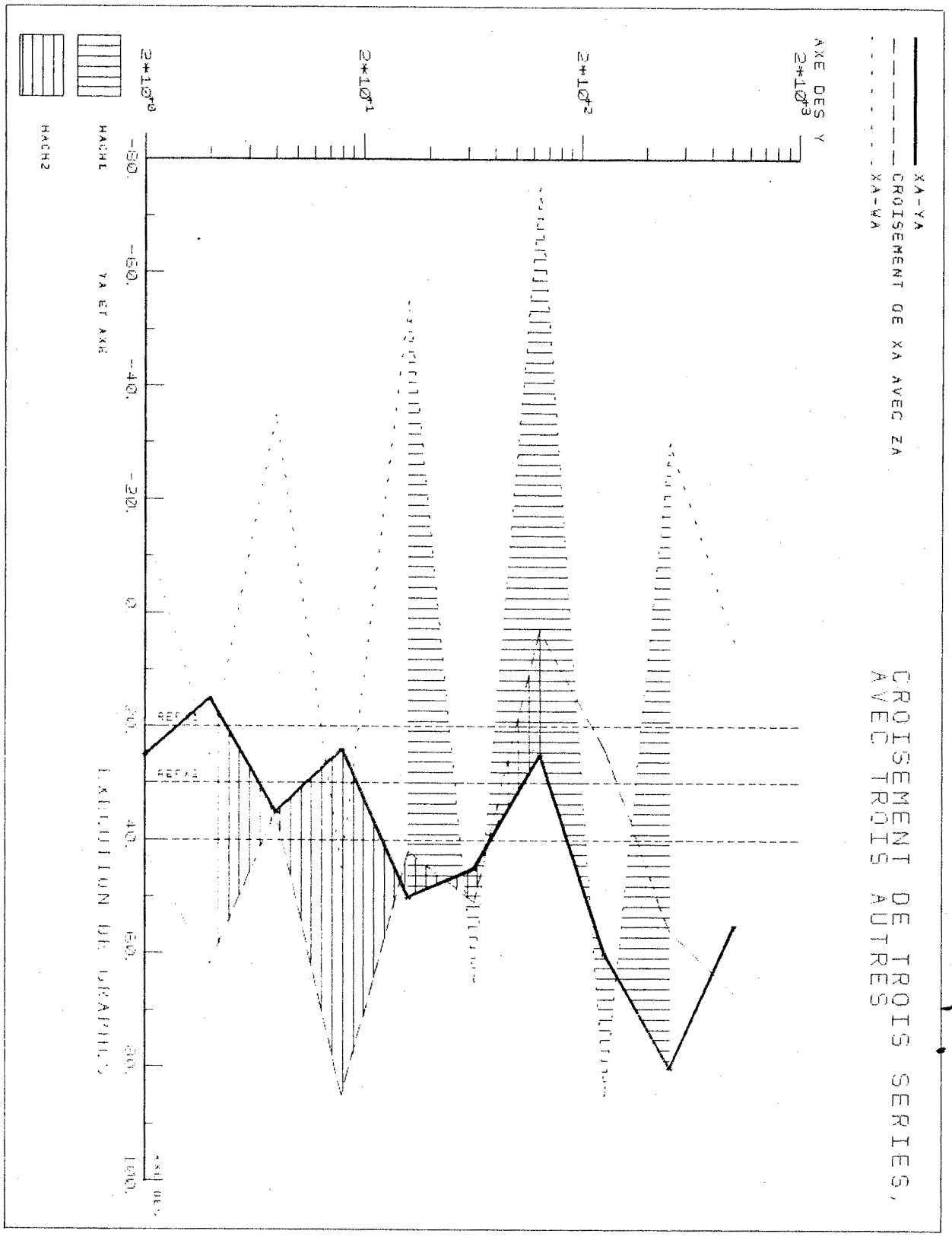
```
<symbolique idpt segobj
/*nom du segment où est stocké le graphique*/>;
<liste hachure de (date dmin;date dmax;entier y1;entier y2)
/* liste hétérogène permettant de définir les zones de hachurages
  chaque liste est constituée de :
   -date début du hachurage
   -date fin du hachurage
   -entier égal à l'ordinal de la première courbe frontière
   -entier égal à l'ordinal de la deuxième courbe
    ( s'il vaut 0 la frontière du hachurage est alors l'axe des temps
   I M P O R T A N T
    les dates sont données dans le calendrier de la 1ière courbe */>;
<liste typeh de (entier numl1;symbolique ident typeh1;
symbolique ident coulh)
/* type de hachurage c'est une liste hétérogène
  chaque élément est constitué de :
   -entier = numéro d'ordre du hachurage
   -ident donnant le type (1ier caractère)
              et l'épaisseur (2ième caractère)
   -ident = couleur
( pour le type on a le choix entre l r h v x
  pour l'épaisseur on peut choisir entre 1 2 3 4 5)*/>;
<reel margb
/*marge basse exprimée en % de la hauteur de la fenêtre*/>;
<reel margh
/*marge haute exprimée en % de la hauteur de la fenêtre*/>;
<reel margd
/*marge droite exprimée en % de la largeur de la fenêtre*/>;
<reel margg
/*marge gauche % de la largeur de la fenêtre*/>;
<liste titecy de chaîne
/*titre de l'axe y , une chaîne par ligne*/>;
<liste titecd de chaîne
/*titre de l'axe x , une chaîne pour une ligne*/>;
<liste titp de (entier numser9;
date dat;chaîne titre1)
/* liste hétérogène qui permet de donner des titres aux pts des
  courbes; chaque élément de la liste est constitué de :
   -entier donnant le numéro de la série concernée
   -date exprimée dans le calendrier de la série
   -chaîne titre du point (15 caractères max )*/>;
```

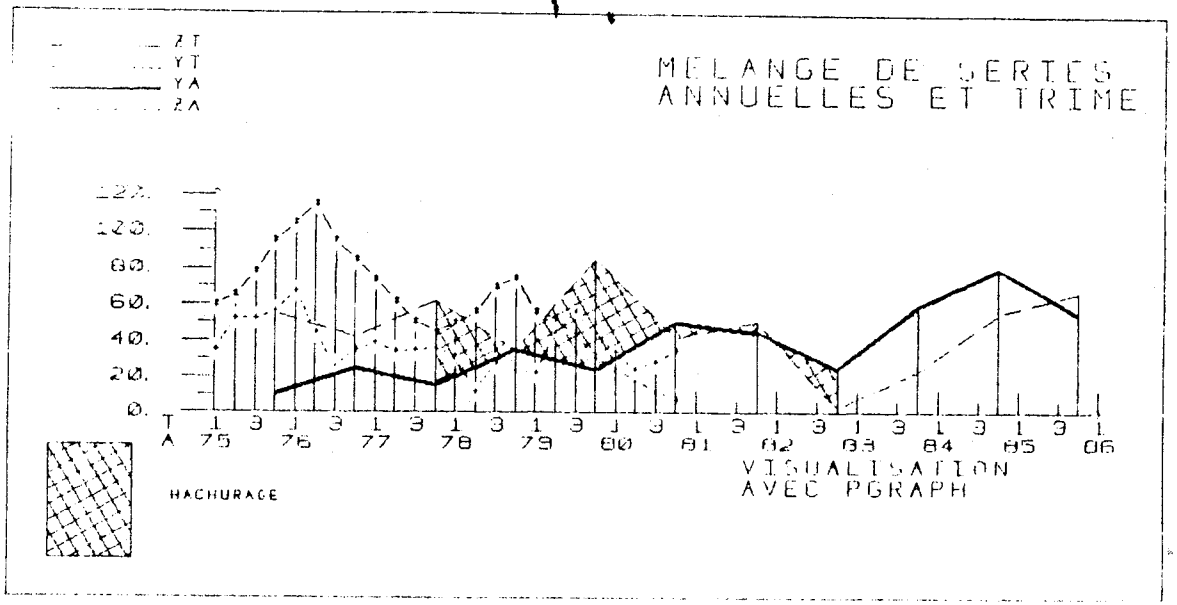
```
<liste titry de (entier numser5;chaîne titre2)
/* liste hétérogène qui permet de légender les courbes
  -entier numéro de la série
  -chaîne titre de la série(40 caractères max)
  si pour une série le titre n'est pas donné c'est le nom de la série
  qui sert de légende*/>;
<liste titrh de (entier numl2;chaîne tith)
/* liste hétérogène qui permet de légender les hachurages
  elle est constituée de :
  -entier = numéro d'ordre du hachurage
  -chaîne = titre du hachurage (40 caractères max) */>;
<liste refo de (date d0;reel yy0;date d1;reel yy1)
/* liste hétérogène; elle permet de définir des droites obliques
  elle est constituée de :
  -date ( abscisse de l'origine du segment )
  -réel ( ordonnée de l'origine du segment )
  -date ( abscisse de l'extremité du segment )
  -réel ( ordonnée de l'extrémité du segment )
  a t t e n t i o n
  les dates doivent être données dans le calendrier fusionné
  à condition que celui-ci soit annuel trimestriel ou mensuel  */>;
<liste titrefo de (entier numl3;chaîne titre3)
/* liste hétérogène pour les titres des références obliques
   -entier = numéro d'ordre de la référence oblique
   -chaîne = titre de la référence (40 c max) */>;
<liste titrefd de (entier numl4;chaîne titre4)
/*  liste hétérogène pour les titres des références verticales
   -entier = numéro d'ordre de la référence verticale
   =chaîne = titre de la référence (40 c max) */>;
<liste titrefy de (entier numl5;chaîne titre5)
/* liste hétérogène pour les titres des références horizontales
   -entier = numéro d'ordre de la référence horizontale
   -chaîne = titre de la référence (40 c max) */>;
<liste police de (symbolique ident nat8;entier numpol)
/* liste hétérogène pour donner des polices aux titres
  elle est constituée de :
  -ident qui indique sur quel titre on veut se placer
  -entier  de 1 à 9 pour le numéro de la police */>;
<liste taille de (symbolique ident nat9;entier ntaille)
/* liste hétérogène pour choisir la taille des caractères des titres
  -ident qui specifie le titre (titg notg....)
```

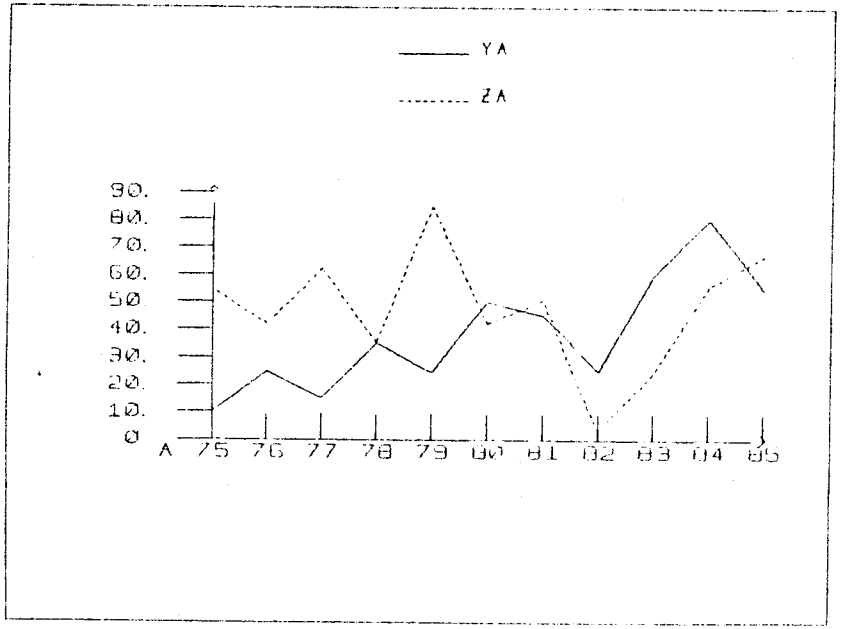
-entier de 1 à 9 */>;
<liste de liste position de symbolique ident
/* liste de liste homogène qui permet de modifier la position
de certains titres comme titg notg titecy titecd
chaque liste est constituée de 2 ou 3 ident
-le 1ier = titg ou notg ou titecy ou titecd
-le 2ième peut valoir gauche droite , haut milieu bas centre
-le 3ième : gauche droite centre haut milieu bas */>;
<liste rot de symbolique ident
/* liste d'idents pour opérer une rotation de 90[sur des titres
avec titecy le titre de l'axe y s'écrit vertical
avec titrefo le titre ne s'écrit plus le long de la référence
avec titrefd le titre s'écrit horizontalement */>;
<liste centrage de symbolique ident
/*un ident par série qui peut valoir gauche , milieu , droite
la valeur par défaut : milieu
si un seul ident est donné il s'applique à toutes les séries*/>)
options=(nvisu,nleg,autod,autoy,autoprd,autopry,trstrd,trsty,slog,
droitey);

Voici quelques exemples d'utilisation de ces commandes.


TROIS SERIES ANNUELLES
YA, ZA ET WA
SERIE YA
ZA
WA
AXE DES VALEURS
100.
80.
60.
40.
20.
0.
-20.
-40.
-60.
-80.
A
75
78
81
84
85
AXE DU TEMPS
ESSAIS DE PARAMETRES
GENERALEMENT PEU UTILISES
MACHINERIE ENTRE YA ET AXE
ZA-TPS
YA-ZA


XA-YA
CROISEMENT DE XA AVEC ZA
XA-WA
AXE DES Y
2*10**0
2*10**1
2*10**2
2*10**3
HACH1
HACH2
YA ET AXE
-60.
-50.
-40.
-20.
0
40.
60.
80.
100.
CROISEMENT DE TROIS SERIES,
AVEC TROIS AUTRES


ZT
YT
YA
ZA
MELANGE DE SERIES
ANNUELLES ET TRIME
140.
120.
80.
60.
40.
20.
0.
T 1 3 J 3 1 3 1 3 1 3 J 3 1 3 1 3 1 3 1 3 1
A 75 76 77 78 79 80 81 82 83 84 85 86
HACHURAGE
VISUALISATION
AVEC PGRAPH


YA
ZA
90.
80.
70.
60.
50.
40.
30.
20.
10.
0
A 75 76 77 78 79 80 81 82 83 84 85

3. La création de commandes

CREERCOMMANDE est la commande de base du logiciel, elle permet son existence. C' est à partir de cette commande que sont créées toutes les autres. On constitue la bibliothèque des commandes standards en appelant pour chacune d'elles CREERCOMMANDE et lui passant un ensemble de paramètres dont voici la description.

```
CREERCOMMANDE      CCOM
CREATION D'UNE COMMANDE
NOMCOM          CHAINE
NOM DE LA COMMANDE A CREER
< MERE          CHAINE  >
NOM DE LA COMMANDE MERE
< ABREV         CHAINE  >
ABREVIATION
< ENTETEPROG      EN-TETE DE PROGRAMME MODULECO  >
ENTETE DE PROGRAMME DECRIVANT LES PARAMETRES
< LISTACCES       LISTE HOMOGENE  SYMBOLIQUE IDENT  >
LISTE D'ACCES
TYPCOM          TYPE DE COMMANDE
TYPE DE LA COMMANDE
```

A l'initialisation du logiciel, lorsqu'on crée l'utilisateur privilégié dont le fichier utilisateur correspond au fichier des commandes standard, on créera cette commande en construisant "à la main" la représentation interne correspondant à CREERCOMMANDE et en appelant par programme la commande elle– même en lui fournissant sa propre description.

Voici quelques exemples de créations de commandes plus ou moins complexes:

```
CCOM 'LIBOUCL' abrev='lbc'
"fonc liste liboucl de symbolique ident
/* liste des variables de bouclage du modele courant */
(<liste libloc de symbolique ident /* liste des blocs */>);"
(tous) foncpasc ;

CCOM 'TAUXOCC' abrev='TOC'  "proc tauxocc
/* impression de la taille et du taux
d'occupation du fichier utilisateur */;
"(tous) typcom=compasc;

CCOM 'CORC' 'estimer' abrev='CORC'
"proc corc
/* estimation par la methode de Cochrane-Orcutt
des equations specifiees par la commande 'ESTIMER'
option sauve    : insertion des equations dans un modele
     nonstock : pas de stockage des coefficients
     nonsort  : pas de sortie au terminal */
(<liste liequa de symbolique ideq
/* Liste d'equations */>;
<date dd /* Date de debut */>;
<date df /* Date de fin */>;
<symbolique ident nomest
/* Nom de l'estimation-par defaut : 'resest' */>;
<symbolique idpt mod /* Nom du modele de sauvegarde */>;
<entier nbiter=20 /* Nombre d'iterations */>;
<reel test=0.005 /* Critere de convergence */>)
options=(sauve,nonstock,nonsort);"(tous) typcom=compasc;
```

Le dernier exemple sera la création d'une commande écrite en langage MODULECO. Dans ce cas le paramètre ENTETEPROG n'a pas à être fourni, tous les renseignements ont déja été donné dans le programme MODULECO.

```
proc 'gestdon' abrev='gdon' typcom=commodl;
```

4. Mise à jour et modifications des commandes

Un ensemble de commandes permettent la destruction, la modification, l'impression et la copie à partir d'un environnement voisin d'une commande. Certaines de ces commandes (MODIFCOM, LISTCOM et EDITCOM que nous avons vu dans un paragraphe précédent) ne s'adressent qu'à des programmes écrits en langage MODULECO. Les autres, c'est à dire DETCOM, CREERCOMMANDE et COPYCOM concernent ausi bien les programmes PASCAL que les programmes MODULECO.

Voici la syntaxe de ces différentes commandes:

Destruction d'une commande:

DETCOM DTC
DESTRUCTION D'UNE COMMANDE
NOMCOM CHAINE
NOM DE LA COMMANDE

Modification d'une commande écrite en langage MODULECO. On peut aussi bien modifier le programme lui-même que les paramètres de la commande.

MODIFCOM MFC
MODIFICATION DE COMMANDE
NOMCOM CHAINE
NOM DE LA COMMANDE

Copie d'une commande appartenant à un autre utilisateur MODULECO.

COPYCOM CPC
COPIE D'UNE COMMANDE PERSONNELLE D'UN AUTRE UTILISATEUR
NOMCOM CHAINE

NOM DE LA COMMANDE A COPIER
< MERE CHAINE >
NOM DE LA COMMANDE MERE
< LISTACC LISTE HOMOGENE SYMBOLIQUE IDENT >
LISTE D'ACCES
UTIL SYMBOLIQUE IDENT
NOM DE L'UTILISATEUR VOISIN

Pour modifier une commande écrite en langage MODULECO, avec les commandes que nous venons de décrire, c'est assez simple. Dans le cas d'une commande en PASCAL, nous distinguerons deux cas: dans le premier cas, on veut remplacer un algorithme existant par un autre, dans le deuxième cas, on veut entiérement revoir la commande et en particulier ses paramètres.

Dans le premier cas, grâce à la modularité du logiciel, on remplace un module PASCAL ou FORTRAN par un autre module du même genre.

Dans le deuxième cas, on sera amené de plus à détruire la commande (DETCOM) pour pouvoir la recréer avec l'ensemble de paramètres et d'options correspondant au nouvel algorithme.

Conclusion

On appelle primitives du langage ces commandes que nous venons de présenter et qui permettent d'ajouter, de supprimer, de modifier ou de consulter une commande. Elles ne diffèrent en rien des autres commandes du langage. Elles forment donc, avec les règles syntaxiques élémentaires du langage, le noyau du langage de commande à partir duquel ce dernier est construit et peut évoluer.

CHAPITRE V

REALISATIONS TECHNIQUES ET CHOIX D'IMPLEMENTATION

1. Représentation interne d'une commande

2. Mémorisation d'une commande

3. La table des commandes

4. L'interpréteur du langage de commande

Tout ce qui concerne l'environnement du langage de commande, c'est à dire la spécification et l'implémentation des primitives qui gèrent les commandes MODULECO a été entiérement à ma charge. Cela comprenait essentiellement la création et la destruction de commandes, l'appel de commandes, l'édition, la modification, l'impression des commandes écrites en langage MODULECO. J'ai contribué à la spécification du langage de commande et ma participation à son élaboration a consisté en l'écriture de l'interpréteur du langage.

J'ai également écrit tout ce qui concerne la gestion des objets et des utilisateurs du logiciels et rédigé le "Manuel Utilisateur" de MODULECO.

Introduction

Les réalisations techniques qui sont décrites dans ce chapitre correspondent aux grandes étapes de la vie d'une commande :

édition, analyse et traduction d'un programme écrit en langage MODULECO.

Le texte du programme est entré en utilisant l'éditeur de MODULECO. Ce texte est ensuite analysé. Si aucune erreur n'est détectée, le programme est traduit en une représentation interne arborescente. Ces actions constituent les trois étapes les plus importantes de la commande EDITCOM qui est utilisée pour programmer en langage MODULECO.

création et mémorisation de la commande

Si on désire conserver ce programme, on le sauvegarde sous forme d'une nouvelle commande du logiciel par appel à CREERCOMMANDE. La commande est alors mémorisée dans le fichier de l'utilisateur. On conserve la table des symboles, l'arbre du code et l'arbre d'appel décrivant les paramètres et les options de la commande.

exécution de la commande

Pour cela il faut l'appeler en lui fournissant un ensemble de paramètres et d'options. A l'aide de la table des commandes, on peut conserver vingt commandes en mémoire. L'interpréteur du langage effectue l'appel dynamique de la commande.

1. Représentation interne d'une commande

Introduction: langage et grammaire

Un *langage de programmation* est un ensemble de mots et de manières de les combiner pour communiquer des instructions à une machine. Deux niveaux extrêmes apparaissent:

celui de l'*expression* qui consiste en une chaîne de caractères

celui du *contenu* qui porte la signification: comprendre c'est trouver l'interprétation correcte du contenu.

On peut donner une définition rigoureuse de l'expression. Un langage possède un répertoire de symboles en nombre fini. Les définitions relatives au niveau de l'expression d'un langage se présentent alors de la manière suivante:

-un *vocabulaire* est un ensemble fini non vide de symboles composés à l'aide des caractères du langage. Les caractères de MODULECO sont les lettres de l'alphabet, les chiffres de 0 à 9 et un ensemble de caractères spéciaux tels que <, ;, %. Les *symboles* sont les entiers, les réels, les mots clés par exemple.

-un *mot* ou encore une *phrase* est une chaîne de longueur finie, composée à l'aide des symboles du vocabulaire.

-on appelera *langage* nimporte quel ensemble de chaînes.

La génération d'un langage est la réalisation d'un processus qui permet de produire toutes les phrases du langage. Le processus de production de chaque phrase doit être fini. La base d'un tel processus est une *grammaire*.

Une grammaire fait intervenir quatre quantités: les symboles terminaux, les symboles auxiliaires, l'axiome et les productions. L'*axiome* est le langage décrit. Les *productions* servent à construire des dérivations. Une *dérivation* consiste à remplacer un symbole auxiliaire par une chaîne au moyen d'une production. A toute dérivation est associée une *arborescence*. Les feuilles sont munies exclusivement des symboles terminaux. Tout symbole terminal ne peut se trouver que sur une feuille. Une grammaire qui produit plus d'un arbre syntaxique pour un programme donné est dite ambigüe. L'analyseur syntaxique, étant donné une grammaire G, balaie la chaîne d'entrée de gauche à droite et produit soit un arbre syntaxique si la chaîne est une phrase du langage engendré par G, soit un message d'erreur dans le cas contraire.

Une grammaire est dite LL(1) si le processus d'analyse balaye la chaîne d'entrée de gauche à droite avec une avance de un symbole seulement pour déterminer la production nécessaire pour construire la dérivation la plus à droite.

Un texte écrit en langage MODULECO soumis à l'analyseur lexical est lu de gauche à droite et caractère par caractère. L'analyseur a pour tâche de reconnaître les symboles du langage. Il construit en mémoire centrale la table des symboles du langage et empile les *fonctions sémantiques* qui serviront à générer la représentation interne de la commande.

1.1 La table des symboles

Elle est organisée en vingt classes. Les symboles sont répartis dans ces classes à l'aide d'une fonction de hachage. A l'intérieur de chaque classe, en cas de collision, les éléments sont rangés par ordre alphabétique. Un élément de table des symboles contient les renseignements suivants:

nom

code haché (entier compris entre 1 et 20)

rang dans la classe

témoin d'initialisation

catégorie du symbole

fonction, procédure, documentation, paramètre
constante, variable, option, nom de commande

taille du paramètre dans le message

taille d'un entier = 1
taille d'un réel = 2
taille d'un booléen = 1
taille d'une date = 2
taille d'une chaîne = 2
taille d'une liste = 2
taille d'un ident = 4
taille d'un symbolique = 2

nom de la variable globale, valeur par défaut

nom du paramètre exclusif suivant

entier pour indiquer le caractère du paramètre

1: paramètre obligatoire sans valeur par défaut
2: paramètre obligatoire avec valeur par défaut
3: paramètre facultatif sans valeur par défaut
4: paramètre facultatif avec valeur par défaut

adresse de la documentation en mémoire centrale

adresse de la documentation dans le membre DOCU

adresse de l'élément suivant dans la classe

valeur de l'élément pour les types élémentaires

ou adresse de la structure contenant la valeur

description des champs pour une liste hétérogène

adresse de la chaîne des options pour une option

Tout symbole apparaissant dans un programme ou dans un en tête de commande (paramètre, variable, constante, option...) sera représenté par un élément de table des symboles qui le décrit complétement.

A l'aide des fonctions sémantiques empilées au cours de l'analyse lexicale, on va pouvoir construire l'arbre n-aire qui représentera le code de la commande ou l'arbre d'appel qui représentera l'en tête de la commande. A l'appel de la commande, l'arbre d'appel sera interprété pour récupérer les paramètres de la commande. Le même interpréteur sera utilisé pour exécuter la commande si elle est écrite en langage MODULECO.

Les fonctions sémantiques les plus importantes sont:

déclaration de procédure ou de fonction, pour initialiser la construction de l'arbre d'appel

déclaration de paramètre liste ou variable élémentaire pour constituer un noeud paramètre défini par l'élément de table des symboles communiqué à la fonction sémantique.

déclaration des options pour créer un noeud option

déclaration du résultat de la fonction pour constituer le noeud résultat qui est le premier paramètre de l'arbre d'appel d'une fonction

appel commande qui permet de communiquer au traducteur l'adresse du noeud racine de l'arbre d'appel de la commande concernée.

fin d'un programme de commande pour terminer l'arbre d'appel de la commande.

1.2 L'arborescence

Une commande est représentée de manière interne par un arbre. Les noeuds terminaux pointent vers des éléments de table des symboles. Les noeuds non terminaux indiquent quel est l'opérateur qu'on va appliquer à l'arbre élémentaire dont ce noeud est la racine. Une commande est donc représentée par une arborescence dont la racine est le noeud non terminal qui représente l'opérateur de la première instruction du programme.

La structure représentant un noeud comporte les informations suivantes:

nom

type

terminal ou non terminal

pour un noeud non terminal

opérateur

adresse du premier fils

nombre de fils

pour un noeud terminal

pointeur vers un élément de table des symboles

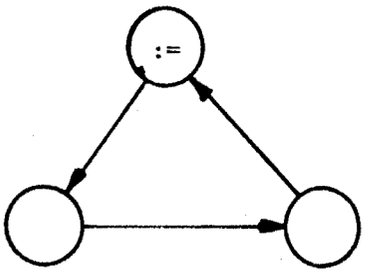
indice d'un champ de liste hétérogène (si besoin est)

nombre d'imbrications pour une liste homogène

table d'adresse d'éléments de table des symboles correspondants aux indices de liste

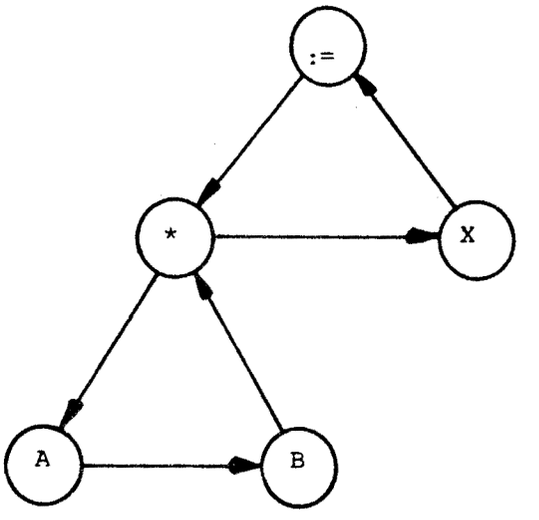
Des exemples de représentation de quelques instructions sont donnés page suivante.

Représentation de l'instruction AFFECTATION



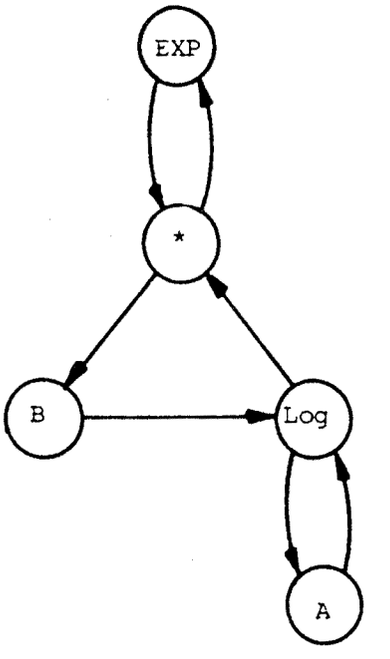
membre droit membre gauche

X := A * B



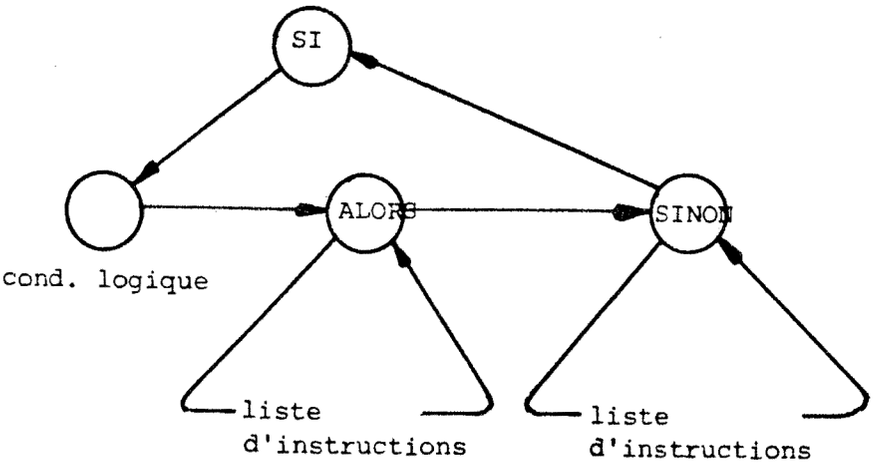
Représentation de puissance

A**B ⇒ exp (B*log A)



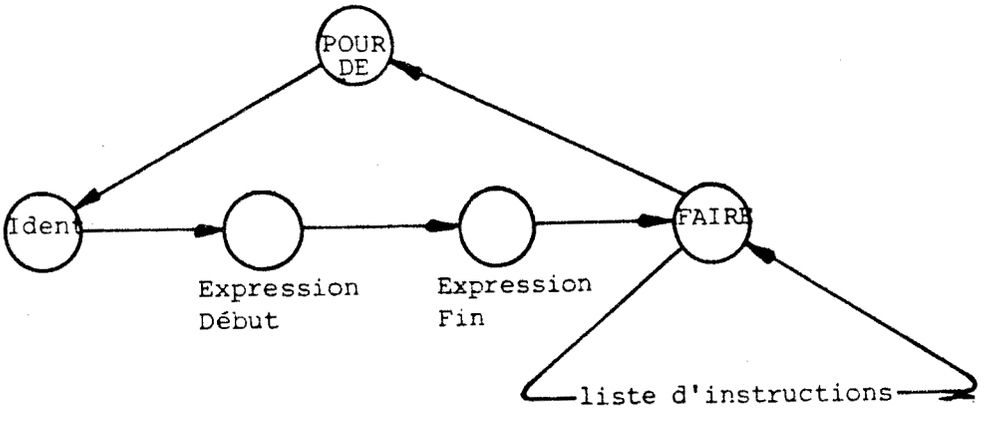
Représentation de l'instruction SI

Si ___ Alors ___Sinon ____

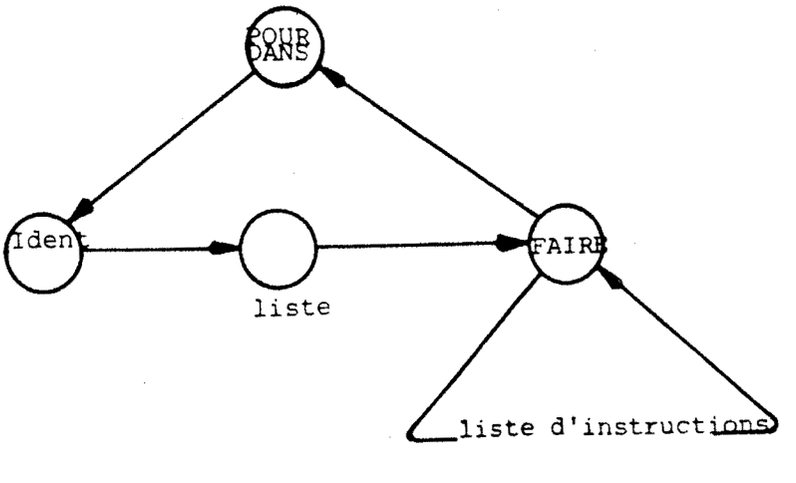


Représentation de l'instruction POUR

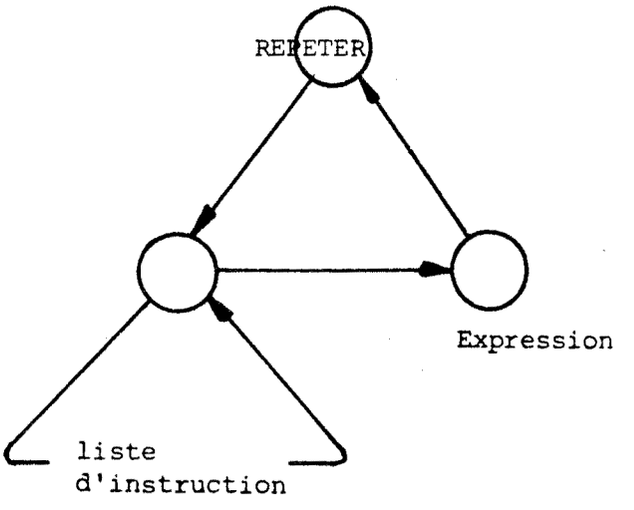
POUR _____ DE _____ A _____ FAIRE _____



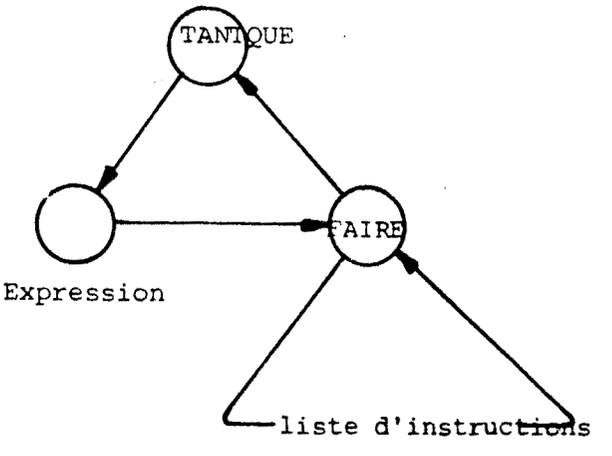
POUR _____ DANS _____ FAIRE _____



Représentation de l'instruction REPETER



Représentation de l'instruction TANTQUE



2. Mémorisation d'une commande

2.1. Structures utilisées

A chaque utilisateur de MODULECO correspond un fichier dans lequel sont sauvegardés tous les objets qu'il a créés, toutes ses données et les résultats de son travail. Le fichier utilisateur est divisé en segment MODULECO. Ces segments constituent le support des divers types d'informations. Ils sont composés de un à dix membres, chaque membre correspondant à une fraction de fichier. Le segment peut être considéré comme un catalogue qui permet de regrouper plusieurs membres ayant un même centre d'intérêt.

Ainsi au lieu d'avoir autant de fichiers que de différents types d'information, l'utilisateur n'a qu'un fichier MODULECO qui constitue l'espace de ses objets.

Lorsqu'on crée un utilisateur, un fichier lui est alloué. Ce fichier est formatté. Quatre segments indispensables à la gestion des objets sont créés: le segment environnement, le segment table des symboles, le segment arbre et le segment calendrier. Seuls les trois premiers nous intéressent.

Le segment environnement

C'est, en somme, l'environnement minimal fourni à l'utilisateur. Il est subdivisé en six membres:

-ENVIR, dans lequel sont stockés tous les descripteurs des objets constituant l'environnement de l'utilisateur.

-BNOM, dans lequel sont stockées les listes d'accés à chaque objet. Celles ci

permettront le partage d'informations en lecture seulement.

-TABLEA, dans lequel sont stockées les tables qui permettent la définition des variables courantes dont nous avons déja parlé.

-DOCU contient toutes les chaînes documentaires.

-DICCO contient les descripteurs des commandes de l'utilisateur.

-CATALOGUE correspond au catalogue des commandes.

La documentation associée à chaque commande et à tous les paramètres de chaque commande est sauvegardée dans le membre DOCU à la création de la commande.

A chaque commande est associé un descripteur qui sera stocké dans le membre DICCO. Ce descripteur est créé au cours de l'exécution de CREERCOMMANDE. Il contient tous les renseignements nécessaires à l'exécution ou à l'appel d'une commande:

son nom

son abréviation (si elle existe)

le nom de sa mère

l'adresse de la documentation

le nombre de ses filles

le nom de la première fille

le nom de la première soeur

son type (fonction, procédure, MODULECO ou non)

adresse de début et de fin de la table des symboles

adresse de début et de fin du code pour une commande écrite en langage MODULECO (0 sinon)

adresse de début et de fin de l'arbre d'appel

Chaque commande est répertoriée une ou deux fois dans le membre CATALOGUE selon qu'elle a ou non une abréviation. Une entrée dans ce membre contient:

le nom de la commande

son abréviation

l'adresse de son descripteur

Pour l'entrée correspondant à l'abréviation les deux premiers champs seront inversés.

Pour stocker ces données, on utilise une fonction de hachage. On fait la somme des ordinaux des deux premiers caractères et des deux derniers caractères du nom de la commande ou de l'abréviation. La fonction rendra un résultat égal au reste de la division de cette somme par le nombre de classes de hachage, augmenté de un. Le fait d'avoir deux entrées distinctes pour la commande et pour l'abréviation et l'utilisation d'une fonction de hachage accélèrent les recherches au moment de l'appel commande. Actuellement, pour une centaine de commandes, nous avons vingt classes d'adressage calculé. Le nombre de collisions reste donc raisonnable.

Le segment table des symboles

A chaque commande est associée une table des symboles. Pour chaque variable de la commande, il existe une entrée dans la table des symboles. On parle aussi d'éléments de table des symboles.

Ce segment est composé de quatre membres:

-CHAMP, où l'on range les descripteurs de champ des listes hétérogènes. Une entrée dans ce membre comprend les renseignements suivants:

nom du champ

type du champ

adresse du champ suivant

-LOCALE contient les éléments de la table des symboles. Une entrée dans ce membre indique:

nom de l'élément

sa catégorie

témoin qui indique qu'une valeur par défaut a été sauvegardée

nom de la variable globale associée

nom du paramètre exclusif suivant

caractère du paramètre (obligatoire ou non, valeur par défaut)

adresse dans le membre BLOCD

adresse de la documentation

pour les paramètres de type élémentaire une valeur éventuellement

pour une liste homogène le nombre de dimensions et le type de éléments

pour une option une adresse dans le membre OPTION

pour une liste hétérogène le nombre de champs et une adresse dans le membre CHAMP

pour une chaîne sa longueur.

-OPTION recoit les blocs décrivant le chaînage en anneaux des options. Une entrée dans ce membre contient les renseignements suivants:

nom de l'option

nom de l'option exclusive suivante

adresse de l'option suivante

-BLOCD où sont stockés les enregistrements correspondant à la valeur des constantes ou à la valeur par défaut système des paramètres. Une entrée dans ce membre indique:

l'adresse du bloc suivant

la valeur de l'élément selon son type

Dans chaque élément de table des symboles, rangé dans le membre LOCALE, on trouve les adresses nécessaires pour rechercher les renseignements sauvegardés dans les autres membres.

Le segment arbre

Aux deux arbres que nous avons déjà décrit correspondent deux membres dans le segment "arbre". Ils ont tous deux la même structure.

-ARBRAPPEL qui recoit les enregistrements décrivant l'arbre d'appel de chaque commande.

-CODE où l'on range l'arbre représentant le code de la commande quand elle est écrite en langage MODULECO.

Un arbre est rangé noeud par noeud. Si c'est un noeud terminal, on conserve les informations permettant de retrouver l'élément de table des symboles qui lui correspond (numéro de classe , déplacement dans la classe). Si c'est un noeud non terminal, on indique l'opérateur représenté et le nombre de fils. Voici la structure d'une entrée dans ces membres:

type du noeud

pour un noeud terminal

code haché de l'élément de table des symboles associé

rang dans la classe d'équivalence

indice d'un champdeliste hétérogène

nombre d'imbrications pour une liste homogène

tableau de noms d'éléments de liste homogène

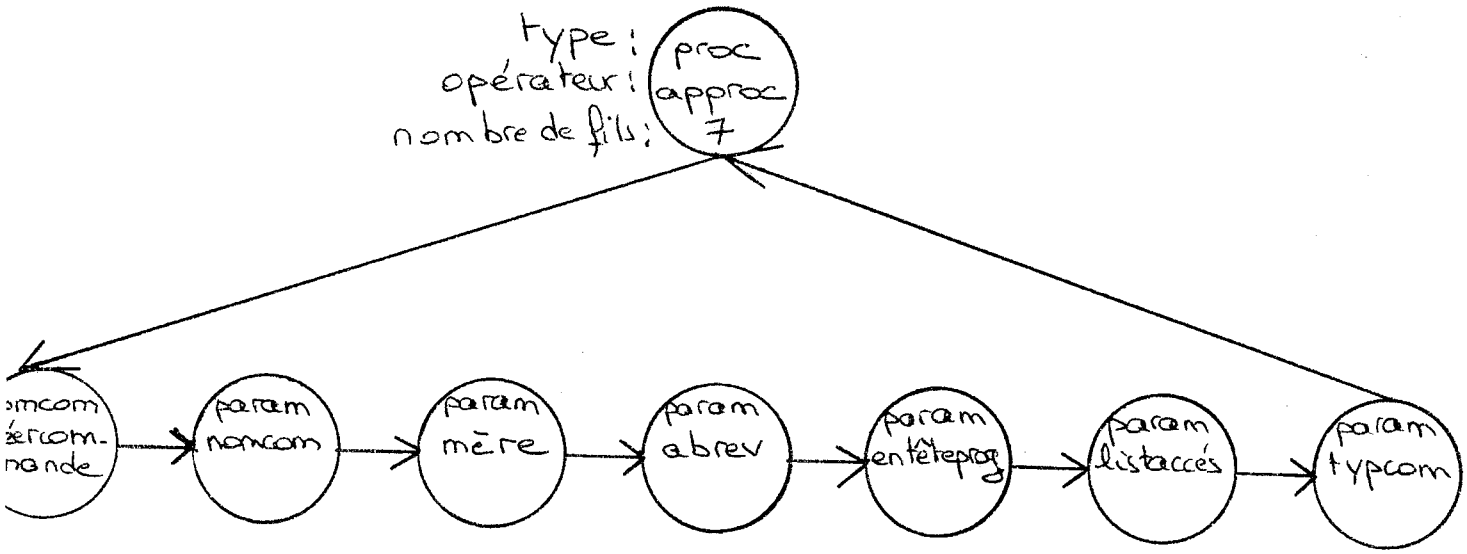
pour un noeud non terminal

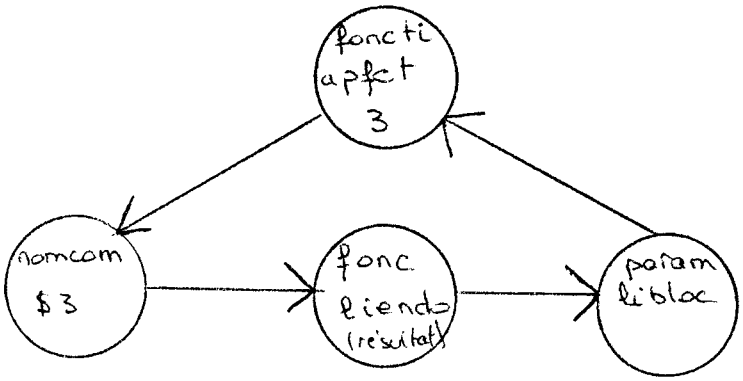
opérateur

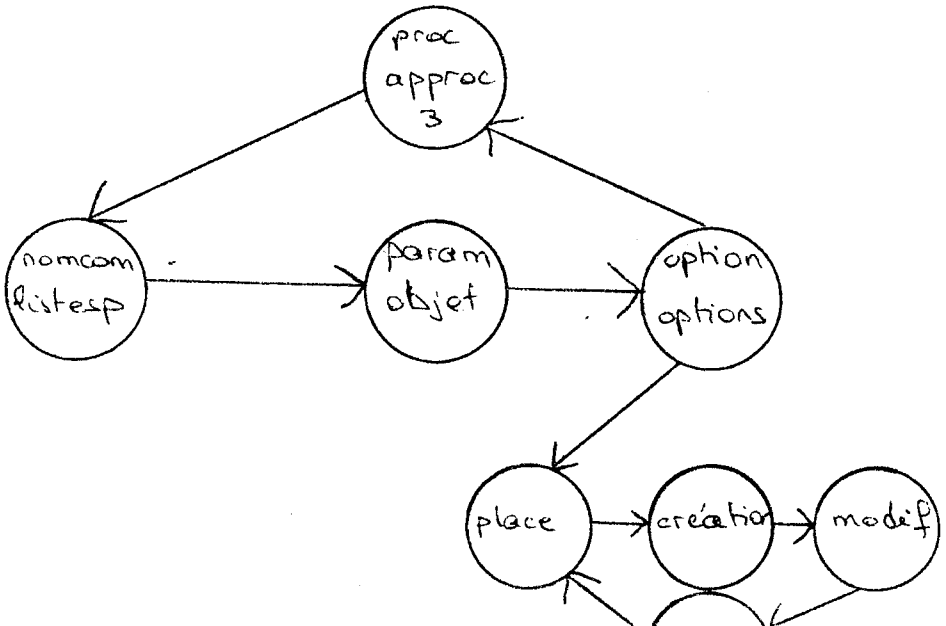
nombre de fils

2.2 Les différentes étapes d'une mémorisation

Au cours de l'analyse du texte de l'appel à CREERCOMMANDE, et plus particulièrement au cours du traitement du paramètre ENTETEPROG, le traducteur a construit l'arbre d'appel de la commande qu'on veut créer. Au sommet, on trouve le noeud "appelcommande" , non terminal, opérateur "approc", ayant un nombre de fils égal au nombre de paramètres plus un. Ce noeud supplémentaire, premier fils, terminal, est le noeud "nom de la commande"; il pointe sur l'élément de table des symboles correspondant au nom de la commande à créer. A première vue, on pourrait penser que ces deux noeuds sont quelque peu redondants, mais en fait leurs rôles sont bien différents. L'un est le père de toute la hiérarchie (noeud non terminal), l'autre est une feuille et ne permet d'avoir accés qu'à l'élément de table des symboles correspondant au nom de la commande. Les autres noeuds sont tous terminaux et pointent sur les éléments de table des symboles correspondants aux différents paramètres de la commande qu'ils représentent. On donne page suivante quelques exemples d'arbres d'appel.


type :
opérateur :
nombre de fils :
proc
approc
7
param
nomcom
param
mère
param
abrev
param
entêteprog
param
listaccès
param
typcom


fonction
apfct
3
nomcom
$3
fonc
liendo
(résultat)
param
liblac


proc
approc
3
nomcom
listesp
param
objet
option
options
place
création
modif

Après avoir calculé à quelle classe d'adressage calculé appartient la commande grâce à la fonction de hachage, on recherche la place où on va effectivement la ranger. Dans une même classe, les commandes sont rangées par ordre alphabétique. On crée et on remplit au fur et à mesure de l'exécution de CREERCOMMANDE les structures qui seront sauvegardés dans CATALOGUE et dans DICCO. Si le paramètre correspondant à la liste d'accès est présent, la liste fournie est sauvegardée dans le membre BNOM et son adresse est conservée dans le descripteur de la commande. Les chaînages sur la mère et sur la première soeur sont mis à jour s'il y a lieu.

L'arbre d'appel est sauvegardé dans le membre correspondant du segment arbre. Pour stocker un arbre, on procède de manière récursive en utilisant deux routines, STONOEUD qui stocke un noeud et STOARBRE qui stocke toute une hiérarchie. Voici en pseudo-pascal l'algorithme de cette sauvegarde :

```
procedure stoarbre (fils, père : pnoeud);

  pour i := 1 to pèreà.nbfils faire
    debut
    stonoeud (fils);
    si fils non terminal alors
      stoarbre (filsà.fils, fils);
    fils := filsà.frere;
    fin pour;
fin stoarbre;
```

La table des symboles de la commande est sauvegardée classe d'équivalence par classe d'équivalence. Pour chaque élément, on crée et on

remplit les différentes structures qui seront écrites dans le fichier de l'utilisateur. La documentation de la commande ou des paramètres est rangée dans le membre DOCU du segment environnement. Selon le type de l'élément, on créera et on sauvegardera des structures complémentaires pour pouvoir retrouver les différents champs d'une liste hétérogène, les valeurs d'une constante ou les valeurs par défaut système des paramètres et la liste des options. Tous les renseignements pour recréer un élément de table des symboles en mémoire sont rangés dans la structure qu'on écrit dans le membre LOCALE du segment de table des symboles ainsi que les adresses de ces structures complémentaires.

Dans le cas d'une commande écrite en langage MODULECO, il reste encore à sauvegarder le code de la commande. Il s'agit encore une fois d'une arborescence de même type que celle de l'arbre d'appel, la différence étant que cet arbre est directement exécutable. On va utiliser les mêmes procédures que pour l'arbre d'appel et on sauvegardera dans le descripteur de la commande l'adresse de début et de fin de l'arbre correspondant au code de la commande.

Une fois toutes ces sauvegardes effectuées, on dispose de toutes les informations nécessaires pour remplir le decripteur de commande. Celui-ci est à son tour sauvegardé dans le membre DICCO du segment environnement. Les deux entrées du catalogue sont constituées, pointant toutes deux sur ce descripteur. L'une correspond au nom de la commande, l'autre à l'abréviation. Elles sont sauvegardées dans le membre CATALOGUE du segment environnement.

La commande est maintenant répertoriée dans le fichier de l'utilisateur. Dans d'autres sessions de travail, on pourra la relire pour pouvoir l'exécuter. Mais au cours d'une même session, on essaie d'éviter les lectures inutiles, trés coûteuses en temps machine.

3. La table des commandes

Pour éviter ces lectures inutiles et coûteuses, on gère en mémoire centrale une table des commandes. Celle-ci permet de conserver en mémoire vingt commandes différentes. Chaque fois qu'on appelle une commande, on cherchera d'abord dans cette table avant d'explorer le fichier des commandes standards ou le fichier de l'utilisateur. On peut espérer ainsi que, au cours d'une session de travail, chaque commande utilisée ne sera recherchée en mémoire secondaire qu'une seule fois. On a pu constater que les économistes ont tendance à utiliser un sous ensemble des commandes mises à leur disposition et toujours les mêmes. Il semble qu'une vingtaine de commandes soient largement suffisantes pour un type de travail donné.

Une entrée dans cette table est composée des champs suivants:

-nom de la commande

-abréviation

-pointeur vers l'arbre d'appel

-pointeur vers le code de la commande

-pointeur vers la table des symboles de la commande

-témoin d'exécution

-booléen indiquant s'il s'agit d'une commande système ou non

-pointeur vers le descripteur de la commande.

Les recherches se feront à l'aide des deux premiers champs, nom et abrev. Avec les trois pointeurs suivants, on dispose de toutes les informations pour appeler et exécuter la commande. Si c'est une commande écrite en

PASCAL, le pointeur vers le code de la commande est mis à nil. Le témoin d'exécution est utilisé lorsqu'on cherche à libérer une place dans la table pour insérer une nouvelle commande et que la table est pleine. On cherchera à écraser une commande qui n'est pas en cours d'exécution.

Le pointeur vers le descripteur permet de trouver les adresses nécessaires au chargement en mémoire centrale de la table des symboles, de l'arbre d'appel et éventuellement du code de la commande. Lorsqu'une commande écrite en langage MODULECO est en cours de construction, ce pointeur vaut nil. En effet, c'est la création de la commande qui lui affectera une valeur. Tant qu'on se contente d'exécuter le programme MODULECO et d'en modifier le texte par appel à EDITCOM, on n'a pas besoin de créer la commande correspondante. Mais pour gérer ce type de programme, on utilise aussi la table des commandes. Lorsqu'on appelle EDITCOM sans paramètre, c'est à dire pour rentrer le texte d'un programme MODULECO, l'analyseur réserve une entrée de la table des commandes dont il peut remplir tous les champs sauf le dernier puisque le descripteur de la commande n'existe pas. Donc on évitera aussi d'écraser une commande dont le dernier champ est à nil parce qu'à ce moment là on serait incapable de retrouver la commande ainsi libérée.

A l'appel d'une commande les recherches se font d'abord dans cette table. Si la commande n'y est pas trouvée elle est cherchée dans le fichier des commandes système puis dans le fichier de l'utilisateur. Si ces recherches sont vaines, un message d'erreur est imprimé et le logiciel attend un autre appel. Lorsqu'on a chargé une commande en mémoire centrale, on crée une entrée dans la table pour y ranger cette commande et ainsi la retrouver rapidement si l'utilisateur la rappelle à nouveau.

4. L'appel d'une commande

Nous venons de voir que la commande sera d'abord cherchée en mémoire

centrale. Si elle n'est pas trouvée dans la table des commandes, on la cherchera alors dans la bibliothèque des commandes standard. L'expérience prouve que les utilisateurs se servent plutôt des outils qui leur sont proposés. Seuls les utilisateurs "avertis" se lancent dans la programmation de nouvelles commandes. Si la commande n'est pas trouvée dans ce fichier, on la cherche alors dans le fichier de l'utilisateur.

Une fois la commande trouvée, il faut vérifier si l'utilisateur a bien le droit de l'appeler dans ce contexte. On utilise à cet effet la pile de contexte dont le sommet détermine le contexte, c'est à dire les commandes qui peuvent être appelées à un instant donné. Si notre commande n'appartient pas à une hiérarchie, on peut l'appeler quand on veut. Si c'est une fille ou une soeur de la commande en sommet de pile alors le contexte est bon.

On empile les adresses des descripteurs de commande. Si la commande appelée est une commande banalisée (n'appartenant pas à une hiérarchie) on n'empile rien; si c'est une soeur de la commande au sommet de la pile, on écrase le sommet de pile avec l'adresse du descripteur de la commande appelée (on n'incrémente pas l'index de pile); si c'est une fille de la commande au sommet de la pile, on incrémente l'index de la pile et on range l'adresse du descripteur de la commande appelée en sommet de pile.

Tant qu'on ne rencontre pas une commande qui fait appel à d'autres commandes, cette seule pile de contexte suffit à déterminer exactement les commandes susceptibles d'être appelées à un instant donné. En effet, rien ne nous dit si la commande en sommet de pile est en cours d'exécution ou est terminée. Or tant qu'elle est en cours d'exécution, on n'a pas le droit d'exécuter une de ses filles. D'où l'utilité d'une pile d'exécution.

A tout instant elle indique quelle est la commande en cours d' exécution. Avant de lancer l'exécution d'une commande, on range l'adresse de son descripteur dans la pile après en avoir incrémenté l'index. Lorsque l'exécution d'une commande est terminée, si c'est une commande banalisée, on décrémente l'index du sommet de la pile d'exécution. Sinon on écrase le sommet de la pile de contexte par celui de la pile d'exécution.

La commande est maintenant en mémoire. Le traducteur parcourt l'arbre d'appel et traite les paramètres un par un. La documentation n'est relue qu'à la demande de l'utilisateur. Toute opération d'entrées-sorties étant relativement chère en temps, on les économise au maximum; c'est pourquoi on ne va chercher la documentation d'une commande ou d'un paramètre qu'à la demande de l'utilisateur. Pour chaque paramètre, on range dans la table des symboles qui lui est associée sa valeur fournie par l'utilisateur. Dans ce cas elle vient écraser sa valeur par défaut système et devient la valeur par défaut utilisateur. Un booléen indique dans la table des symboles si le paramètre a été affecté ou non.

L'arbre d'appel est rempli. On voudrait maintenant pouvoir appelé n'importe quelle commande ayant un certain nombre de paramètres de types variés en utilisant un dispositif commun. On cherche, en somme, un moyen de communication entre le logiciel et chaque commande. On utilise une structure particulière, qu'on a appelé message parce qu'il transmet des informations. Un message est constitué d'une partie commune à toutes les commandes, l'en tête. Il indique le nom de la commande, un masque pour connaître les paramètres présents à l'appel, un masque pour connaître les options présentes à l'appel et un code d'erreur. Derrière cet entête, on trouve un tableau qui contient tous les paramètres de la commande. Pour la commande les champs de cet structure correspondent au type des différents paramètres qu'elle attend. On utilise ici les facilités de compilation séparée. La commande reètre une structure contenant outre un en tête standard, tous les paramètres qu'elle attend. Le programme qui remplit ce message, ne connaît pas le type des différents paramètres de la commande; il remplit, lui, un tableau d'entiers.

Ainsi une commande quelqu'elle soit et en particulier quelque soit le nombre et le type de ses paramètres est appelée avec comme seul paramètre l'adresse d'un message qui contient toutes les informations nécessaires à son déroulement. On réalise de cette manière un appel tout à fait standard et indépendant de la commande dont on veut lancer l'exécution. C'est l'interpréteur du langage de commande qui est chargé de remplir le message et d'appeler la commande.

5. L'interpréteur du langage de commande

Un langage de programmation peut être implémenté à l'aide d'un compilateur, d'un interpréteur ou d'un dérivé des deux produits précédents.

Un compilateur permet de transformer un programme écrit en langage source en un programme formulé en un langage proche du langage machine ou langage d'assemblage. Un objet exécutable est généré. Une phase de compilation est donc rajoutée au traitement du programme et chaque fois que l'on fait la moindre modification dans le programme il faut le recompiler. De plus l'écriture d'un compilateur est une opération lourde à programmer. L'avantage est que la génération d'un code exécutable permet d'aller trés vite à l'exécution du programme.

Pour un programme interprété, les instructions sont exécutées aussi vite que possible dés qu'elles ont été reconnues. Le résultat produit par un interpréteur n'est pas un programme ni un module-objet mais le résultat de l'exécution du programme. On supprime donc la phase de génération de code exécutable mais par contre l'exécution du programme est beaucoup plus longue. Pour écrire un interpréteur on peut utiliser un langage de haut niveau ce qui a l'avantage de faciliter le comprehension, la mise au point, la maintenance et le développement du programme. L'interpréteur est totalement indépendant du système d'exploitation et de l'ordinateur utilisé ce qui n'est pas le cas du compilateur. Donc on évite les problèmes liés au système ce qui nuirait à la portabilité du produit. Enfin l'interprétation d'un programme permet un meilleur contrôle de l'exécution et notamment les récupérations d'erreurs.

La représentation interne d'un programme est généré par un traducteur à deux passes: l'analyse syntaxique et l'analyse sémantique plus la traduction. La première passe construit une pile de fonctions sémantiques et la deuxième à partir de cette pile, construit l'arbre n-aire représentant le programme. On n'a

pas besoin de conserver le source du programme écrit en langage MODULECO. Un désassembleur a été implémenté qui, à partir de l'arborescence représentant le programme, est capable d'en recréer le source.

Voici un exemple de travail du désassembleur. C'est le source du programme de la commande GESTDON que nous avons créée précédemment. Nous l'avons obtenu grâce à la commande LISTCOM qui appelle le désassembleur pour recréer le source de la commande qu'on lui demande d'imprimer.

```
PROC GESTDON é GESTION DE PARAMETRES è;
DATE D2,D1,DINTER;
SYMBOLIQUE IDENT IDPER;
SYMBOLIQUE IDPT S1;
 ECRIRELG 'NOM DU SEGMENT DE TRAVAIL ?';
 LIRE S1;
 ECRIRELG 'CALENDRIER';
 LIRE IDPER;
 ECRIRELG 'DATE DE DEBUT ?';
 LIRE D1;
 ECRIRELG 'DATE DE FIN ?';
 LIRE D2;
 SI D2<D1 ALORS
   DINTER := D1;
   D1 := D2;
   D2 := DINTER;
 FIN ;
 INITSEG S1 D1 D2 IDPER ;
FIN GESTDON ;
```

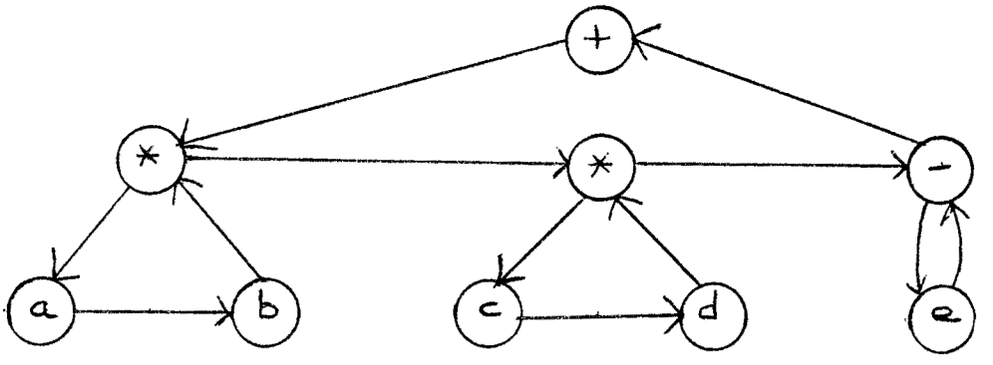
L'interpréteur parcourt l'arbre du code de la commande en préfixé. Par exemple l'expression

(a * b) + (c * d) - e

sera parcouru ainsi

+ * a b * c d - e

Voici l' arbre correspondant



Etant donné un noeud opérateur, on parcourt la chaîne fils de gauche à droite. Quand on rencontre un noeud terminal, on récupère sa valeur et on passe au suivant. Quand on rencontre un noeud non terminal ou noeud opérateur, on traite le sous-arbre dont il est le père en rappelant cette procédure qu'on est en train d'exécuter. On utilise la récursivité du langage PASCAL.

L'interpréteur se compose essentiellement d'une procédure, EVAL, qui évalue un noeud et renvoie le résultat. Il a, en somme, la structure de l'instruction CHOIX de PASCAL, les clauses du choix étant les différents opérateurs du langage MODULECO. Ceci est valable pour les noeuds non terminaux. Pour les noeuds terminaux le résultat de l'évaluation est la valeur contenue dans l'élément de table des symboles indiqué par le noeud. Voici, écrit en pseudo-PASCAL le squelette de cette procédure.

```
procedure eval (p : pnoeud;  /* noeud à évaluer */
          var res : pbloc); /* resultat de l'évaluation */

si p non terminal alors
  q := pà.pfils;
  choix pà.operateur entre
    plus:
```

```
    moins:

    mult:

    divise:

    convr:

    convi:

    affect:
      eval (q, res);
      q := qâ.frere;
      ranger_resultat (q, res);

    approc:



    lirelg:

  sinon
    res := tsâ.valeur;

fin eval;
```

En fait l'interpréteur contient deux procédures importantes. Nous venons d'en voir une, EVAL. L'autre s'appelle EVALP et sert à l'appel des commandes écrites en PASCAL, que ces appels soient des instructions d'une commande écrite en MODULECO ou soient directement tapée à la console. Le rôle principal de EVALP est de remplir le message qui sera passé en paramètre à la commande. L'interpréteur considère le message comme étant composé d'un en-tête et d'un tableau d'entiers. Il va parcourir l'arbre d'appel de la commande et remplir le message à partir des informations qu'il trouvera dans cet arborescence. Pour tous les fils de type PARAM, on regarde la valeur du témoin d'initialisation. Si le paramètre a été fourni par l'utilisateur, on positionne dans le masque des paramètres présents le bit correspondant à ce paramètre (premier bit, premier paramètre....ième bit, ième paramètre) et on remplit le message avec la valeur contenue dans l'élément de table des symboles adressé par ce noeud. Pour les fils de type OPT (options), si le témoin d'initialisation a pour valeur vrai , alors

on positionne le bit correspondant dans le masque des options présentes.

Une fois le message rempli, l'interpréteur réalise l'appel dynamique de la procédure de même nom que la commande, en lui passant en paramètre l'adresse du message.

Lorsque le code de la commande appelée est écrit en langage MODULECO, le contrôle est donnée directement à la première instruction du programme qui est interprétée par appel à la procédure EVAL.

CONCLUSION

Le langage de commande réalisé permet au logiciel MODULECO d'approcher du logiciel idéal de modélisation économétrique décrit au chapitre II et dont nous rappelons les caractéristiques.

-commodité et souplesse d'emploi: une aide interactive est fournie à tout moment à l'utilisateur grâce à la documentation associée à chaque commande et à chaque paramètre. La notion de contexte de travail avec la hiérarchie des commandes évite à l'utilisateur des appels de commande inopportuns.

-simplicité d'utilisation: l'apprentissage de MODULECO est rapide; il ne requiert aucune connaissance informatique. Le caractère facultatif de la plupart des paramètres et les valeurs par défaut rendent le logiciel trés facile à utiliser.

-possibilité d'extension: comme nous l'avons montré au chapitre IV, cet objectif a été pleinement atteint. On a pu rajouté sans problèmes les commandes graphiques qui ont une quarantaine de paramètres et une dizaine d'options. Tous les types du langage MODULECO y sont représentés et bien souvent sous leur forme la plus complexe.

-mode interactif ou traitement par lot: ces deux modes de travail sont offerts à l'utilisateur.

intégration du système: on peut passer facilement d'une tâche à une autre, le langage de commande servant de lien entre tous les sous-systèmes qui composent MODULECO.

complétude: plusieurs méthodes d'estimation et de simulation sont fournies.

Un logiciel conversationnel et intégré est satisfaisant pour les utilisateurs. Mais si l'on essaie d'être un peu plus critique, on s'aperçoit que cette interaction Homme-Machine est nécessaire. En effet le système ne saurait pas se débrouiller tout seul si l'utilisateur n'était pas là pour lui souffler ce qu'il doit faire.

Par exemple, pour pouvoir utiliser en simulation les observations sur le passé de chaque variable apparaissant dans un modèle, il faut transposer le segment de séries correspondant. Cette opération reste à la charge de l'utilisateur. On peut trouver un autre exemple de cette faiblesse du logiciel dans le sous-système d'estimation. Chaque méthode fournit un grand nombre de résultats et de statistiques mais aucune aide à l'interprétation. Il n'y a pas non plus d'aide pour le choix des méthodes.

Dans un logiciel de modélisation, on cherche à augmenter le nombre de méthodes disponibles pour une tâche donnée. La richesse en méthodes est ainsi devenue un critère fondamental d'évaluation. On cherche également à élargir les fonctionnalités du logiciel en regroupant des tâches de nature différentes mais complémentaires. Le fait d'améliorer la facilité et le confort d'utilisation du logiciel, le rendent plus accessible et en particulier à des utilisateurs qui ne possèdent pas forcément la connaissance et l'expertise voulues. Il en résulte deux attitudes prédominantes:

soit l'utilisateur emploie systématiquement la méthode la plus élaborée ou la plus générale pour une tâche donnée,

soit il se limite à l'emploi de quelques commandes simples qu'il a appris à bien connaître.

Dans tous les cas le logiciel est sous-utilisé. Au pire, il fournit des résultats faux sans s'en apercevoir.

En reprenant le cas de l'estimation, il serait souhaitable d'une part que le logiciel soit capable de détecter qu'une méthode est inapplicable compte tenu du contexte et d'autre part qu'il puisse indiquer la méthode la plus appropriée. Il faudrait en outre qu'il soit capable de justifier ses choix et ses décisions.

Il paraît donc extrêmement intéressant d'incorporer à un logiciel tel que MODULECO l'expérience du spécialiste en économètrie. Une approche de type intelligence artificielle permettrait d'offrir aux utilisateurs une aide beaucoup plus efficace et une connaissance liée aux méthodes offertes, aux objets manipulés et au domaine d'application.

BIBLIOGRAPHIE

Brown P.J.
"Writing interactives compilers and interpreters"
Ed. Wiley and Sons 1979

E.P.S
Reference Manual
Data Resources Inc Juin 1978

E.P.S
Techniques
Data Resources Inc Janvier 1977

Eisenpress Harry
"An Ideal Econometric Computer Program"
I.B.M Corporate Headquarters, Armonk, N.Y., 1975
Doc Roneo (8 pages)

Fouquet D.
"Problèmes informatiques généraux et gestion de données"
Méthodes mathématiques de la modélisation macroéconomique
Ed. P.Malgrange
Synthèses du SESORI Mai 1979

Jorrand P.
"Contribution au développement des langages extensibles"
Thèse d'état I.N.P.G 1974

Kogiku K.C
"Introduction aux modèles macro-économiques"
Editions Sirey, 1971

Kupka I., Wilsing N.
"Conversationnal Langages"
Ed. Wiley and Sons 1980

Lafay J.D
"Ajustement de modèles macroéconomiques simples
sur les données francaises"
(1950-1968)
Revue d'Economie Politique 1972 pp1135-1171

Lesourne J.
"Cours d'économie et statistique industrielles"
Tome 1 et 2
Conservatoire National des Arts et Métiers
Support du cours de D.E.S.T, valeur complémentaire

Malinvaud E.
"Méthodes satistiques de l'économétrie"
Ed. Dunod

Mazier J
"La macroéconomie appliquée"
Ed. PUF 1978

Nepomiastchy P. et Rechenmann F.
"The Equation Writing External Language of the Moduleco Software"
Journal of Economic Dynamics and Control 5 (1983) North-Holland
pp37-57

Oudet B. et Nepomiastchy P
"Le projet MODULECO"
Rapport Interne INRIA 1980

Oudet B. et Ruderman G.

"Etude comparative des logiciels disponibles"
Méthodes mathématiques de la modélisation macroéconomique
Ed. P.Malgrange
Synthèses du SESORI, Mai 1979

Oudet B.
"Un système interactif d'aide à la décision
dans un jeu d'entreprise"
R.A.I.R.O Informatique 1978 Volume 12 no 3 pp 233-245

Pincemaille B.
Manuel de Référence du logiciel Xing
Notice d'emploi de Apache
Direction de la Prévision du Ministère de l'Economie et des Finances

Rechenmann F
"Intelligence Artificielle et construction de modéles dynamiques"
Symposium-Exposition Intelligence Artificielle et Productique
20-22 Nov 1984 Paris

TROLL
Computer Operating Activity
Troll Reference Manual (Standard System)
National Bureau of Economic Research Inc
Cambridge Massachussetts (1975)

T.S.P
User's Manual Third Edition May 1973
Computer Sciences Corporation
El Segundo California

Vauquois B
"Calculabilité des langages"
Logique et programmation. C3 Maîtrise d'informatique
Support de cours polycopié

Vignard P
"Système interactif extensible d'aide à la modélisation
et expertise économétrique"
Rapport de DEA INPG Juin 1983

Documentation Interne MODULECO

Rechenmann F
"Manuel d'Introduction"

Hamoniaux-Lespinasse P
"Manuel Utilisateur"

Gardien R, Hamoniaux-Lespinasse P
"Edition, création et appel de commandes"

Hamoniaux-Lespinasse P
"Au sujet des paramètres"

Dronsart J, Morinière M
"Les commandes graphiques de MODULECO"